The Dissertation Committee for Samuel Isaac Ward
certifies that this is the approved version of the following dissertation:

Physical Design Automation of Structured High-Performance Integrated Circuits

Committee:

David Z. Pan, Supervisor

Charles Alpert

Adnan Aziz

Michael E. Orshansky

Earl Swartzlander

Physical Design Automation of Structured High-Performance Integrated Circuits

by

Samuel Isaac Ward, B.S.E.E, M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2013

Dedicated to my wife Britt, and my children Caleb, Hailey, McKenna and Raegan and my loving family members who were always there to support me.

Acknowledgments

First off I would like to thank my dad, mom and grandparents. Without their love, support and encouragement throughout the years I would not have the courage or ability to press forward. I also want to thank my wife and children for all the times they sacrificed their time to allow me to continue school. And for my mentor, Devorah Brock, thank you for being patient and bearing the brunt of my frustrations over the years. To all of you, I am truly grateful.

I want to thank each of my committee members for their time and feedback. Each of you has been instrumental in encouraging and motivating me to continue. Your passion for excellence in all you do has impacted me greatly. In particular, I would like to thank Professor Earl Swartzlander for seeing potential in me early on and encouraging me to continue pursuing school. Without your mentorship and technical feedback, I would not have continued or be where I am today. I would like to thank Professors Adnan Aziz and Michael E. Orshansky for your recommendations that helped continue school and additionally, your technical feedback has been invaluable throughout this work. Finally I would like to thank Dr. Charles Alpert. You truly have a gift for leading and empowering young researchers and none of this would have been possible without your help and support. I count you each as great

mentors and friends.

Finally to my Advisor David Z. Pan, you have truly embodied the university motto of What starts here, changes the world through great teaching, unyielding passion and vision. I thank you for all the feedback, advice and friendship throughout the years. You truly made this work possible and I have the highest respect for you.

Acknowledging and thanking everyone that has helped and encouraged me along the way is not possible but certainly to all my IBM colleagues, each of you Are amazing friends and researchers that helped refine me daily. Nat Viswanathan and Zhuo Li, for all those late night hours we spent brainstorming, editing and rewording, I am deeply indebted. Additionally, Myung-Chul Kim your work is truely groundbreaking and opened the doors to many of these advancements. Lastly, Duo Ding and other UTDA members, thank you for all the help and advice throughout the years. Your input and contributions were indispensable.

Physical Design Automation of Structured High-Performance Integrated Circuits

Publication No. _____

Samuel Isaac Ward, Ph.D.
The University of Texas at Austin, 2013

Supervisor: David Z. Pan

During the last forty years, advancements have pushed state-of-the-art placers to impressive performance placing modern multimillion gate designs in under an hour. Wide industry adoption of the analytical framework indicates the quality of these approaches. However, modern designs present significant challenges to address the multi objective requirements for multi GHz designs. As devices continue to scale, wires become more resistive and power constraints significantly dampen performance gains, continued improvement in placement quality is necessary. Additionally, placement has become more challenging with the integration of multi-objective constraints such as routability, timing and reliability. These constraints intensify the challenge of producing quality placement solutions and must be handled carefully. Exasperating the issue, shrinking schedules and budgets are requiring increased automation by blurring the boundary between manual and automated placement. An example

of this new *hybrid* design style is the integration of structured placement constraints within traditional ASIC style circuit structures.

Structure aware placement is a significant challenge to modern high performance physical design flows. The goal of this dissertation is to develop enhancements to state-of-the-art placement flows overcoming inadequacies for structured circuits. A key observation is that specific structures exist where modern analytical placement frameworks significantly underperform. Accurately measuring suboptimality of a particular placement solution however is very challenging. As such, this work begins by designing a series of structured placement benchmarks. Generating placement for the benchmarks manually offers the opportunity to accurately quantify placer performance. Then, the latest generation of academic placers is compared to evaluate how the placers performed for these design styles. Results of this work lead to discoveries in three key aspects of modern physical design flows.

Datapath placement is the first aspect to be examined. This work narrows the focus to specifically target datapath style circuits that contain high fanout nets. As the datapath benchmarks showed, these high fanout nets misdirect analytical placement flows. To effectively handle these circuit styles, this work proposes a new unified placement flow that simultaneously places random-logic and datapath cells. The flow is built on top of a leading academic force-directed placer and significantly improves the quality of datapath placement while leveraging the speed and flexibility of existing algorithms.

Effectively placing these circuits is not enough because in modern high

performance designs, datapath circuits are often embedded within a larger ASIC style circuit and thus are unknown. As such, the next aspect of structured placement applies novel data learning techniques to train, predict, and evaluate potential structured circuits. Extracted circuits are mapped to groups that are aligned and simultaneously placed with random logic.

The third aspect that can be enhanced with improved structured placement impacts local clock tree synthesis. Performance and power requirements for multi-GHz microprocessors necessitate the use of a grid-based clock network methodology, wherein a global clock grid is overlaid on the entire die area followed by local buffered clock trees. This clock mesh methodology is driven by three key reasons: First, full trees do not offer enough performance for modern microprocessors. Second, clock trees offer significant power savings over full clock meshes. Third, local clock trees reduce the local clock wiring demands compared to full meshes at lower level metal layers. To meet these demands, a shift in latch placement methodology is proposed by using structured placement templates. Placement configurations are identified a priori with significantly lower capacitance and the solutions are developed into placement templates.

Results through careful experimentation demonstrate the effectiveness of these approaches and the impact potential for modern high-speed designs.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern VLSI environments rely heavily on computer aided design (CAD) tools to quickly and effectively build state of the art circuits [3, 7, 13, 64, 57, 11]. Advances in high level synthesis techniques [51, 52] enable design teams to verify and build billions of transistors on a single chip [75, 96]. Standardizations such as fixed size standard cell libraries [1] and advances in physical modeling [101, 63] enable automated tools to handle much of the work once done manually. Modern optimizations often generate solutions better than manual results could achieve at much smaller turnaround time. This results in a reduction in cost and increase in performance for most of the logic function.

Recent gains in placement techniques have been impressive [53, 81, 2] yet the modern placement has become significantly more challenging with the integration of multi-objective constraints such as routability, timing and reliability directly into the placement objective. These constraints intensify the challenge of producing quality placement solutions and must be handled carefully. Exasperating the issue, shrinking schedules and budgets are requiring increased automation by blurring the boundary between manual and auto-

mated placement.

Structured constraints are one example of the emerging placement landscape [92]. The specific constraints vary widely and cover many circuit functions including datapath placement, staging large bus signals, clock synthesis, design for test, and queue design.

Modern automation flows are broken into a series of eight performance optimization transforms as shown in Figure 1.1. This work examines three aspects of the structured constraint problem, shown in green in Figure 1.1, including integrating structured constraints into analytical placement flows, extraction of structured circuits and developing structured latch placement templates.

To begin, it is important to gain a fundamental understanding why analytical placement techniques do not perform well on specific circuit structures. One circuit style where structure is often present is within datapath functions. Additionally, this is an area lacking widespread placement automation. Datapath aware placement has been a controversial topic for many years [55, 31, 54]. The fact remains, industrial designs require manual place and route intervention to meet modern high-performance design constraints. Though resent published works do exist [90, 14], the lack of widespread industrial adoption and continued interest indicates the challenging nature of this problem.

There are two major reasons that a solution to the datapath problem does not exist. First is the nebulous definition of what is "datapath" makes

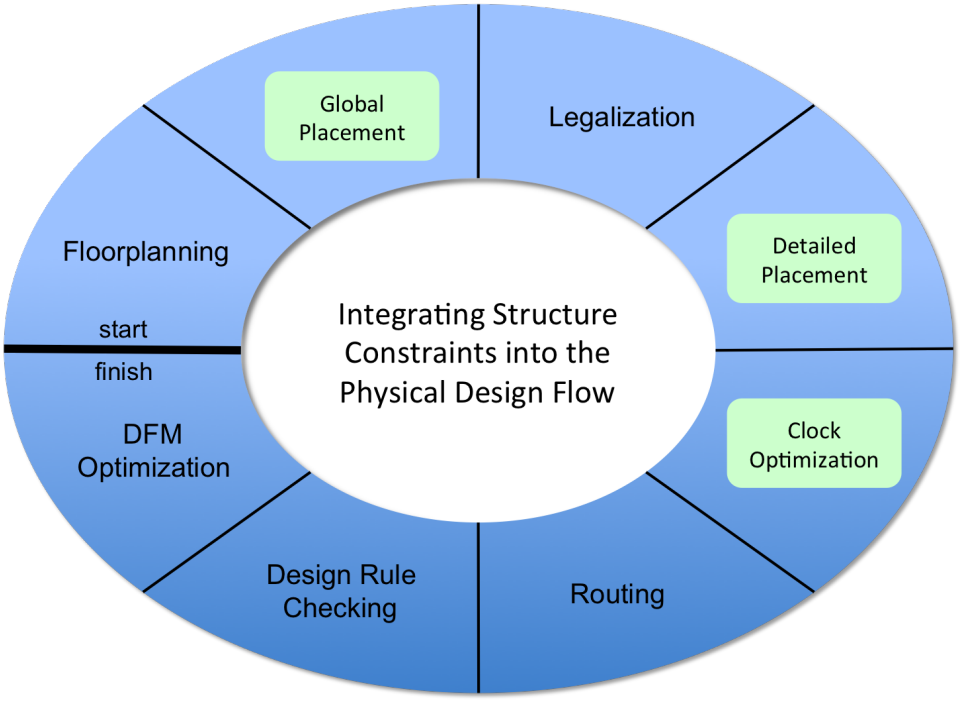


Figure 1.1: Modern physical design flows consist of eight stages starting with floorplanning and ending with design for manufacturing (DFM) optimizations. This work integrates structured constraints into three key stages, global placement, detailed placement, and clock optimization.

datapath automation a moving target. Frequently, any logic style requiring manual design effort and some form of regularity gets lumped into the category of "datapath". Whether logical, hierarchical, physical, arithmetic, or other, the ambiguous definition of datapath logic frustrates the efforts for full automation. This results in impression that a particular automated solution works for a subset of designs but does not work in the general "datapath" case. Second, though much progress has been made in simultaneously balancing multiple constraints such as timing, routing, or power, datapath style

logic often requires careful planning to generate solutions meeting all of these requirements. Frequently in practice automated place and route tools generate a solution meeting timing constraints, but have no hope of routing or vice versa a routable solution that does not meet timing requirements.

In spite of this, accurately measuring suboptimality of a particular placement solution is very challenging. To shed light on exactly why placers have such a hard time with structured circuits, this work begins by designing a series of structured placement benchmarks. Generating manual placement for the benchmarks offers the opportunity to accurately quantify placer performance and examine where the placers fall short.

In Chapter 2, custom constructed structured datapath designs with hand-designed layouts are presented. The degree of suboptimality when compared to manual placement is then quantified for these design styles. The benchmarks consist of two custom designed circuit structures, a Rotate circuit and an Enable circuit are described in Sections 2.2 and 2.3, respectively. From this, sixteen benchmarks are developed with varying degrees of whitespace. with results presented in Section 2.4. Results of this work lead to discoveries in three key aspects of modern physical design flows.

The first key discovery is that analytical placement of high fanout nets disrupt structured circuits causing significant routed wirelength degradation. Additionally, these structures are often deeply embedded within ASIC style logic circuits increasing the complexity for minimizing global wirelength. Thus, a single unified placement flow handling both structured and random logic

simultaneously is extremely valuable, improving design time, solution quality, and saving development and maintenance costs.

Chapter 3 shows that with careful design guidance on high fanout nets, existing Half-Perimeter Wirelength (HPWL) driven placers can better handle designs with embedded structured logic. It is accomplished via a set of effective placement techniques amenable to incorporation within existing random-logic placers. The flow leverages the speed and flexibility of state-of-the-art HPWL-driven placers, while imposing alignment constraints[1] to achieve better regularity and Steiner Wirelength (StWL).

Section 3.2 outlines a key issue with current academic placers, namely the inadequacies and specifically the lack of fidelity of the HPWL model versus the StWL model when evaluating and placing structured logic. Additionally, this section provides a fundamental insight to alignment: alignment of the datapath in either direction guides indirect StWL optimization, and significantly improves total StWL and routing congestion.

Next, a novel placement flow, referred to as Structure-Aware Placement Techniques (SAPT), is developed in Section 3.4. The flow can be incorporated within existing HPWL-driven placers to enable better alignment of the structured nets or datapath cells during both global, Section 3.5 and detailed placement Section 3.6. The effectiveness in both total wirelength and routing congestion is demonstrated in Section 3.7.

[1] Alignment constraint was also discussed in [34] as an example of geometric constraint handling, but no circuit structures were considered.

The second key discovery is that it is possible to characterize and extract specific structures that placers perform poorly on. In the era of modern design where structured circuits are embedded within a single large ASIC style circuit, automatically identifying the structured portions of a design is critical. Structured or datapath extraction techniques in the past generally focused on functional or structural levels. Functional regularity extraction identifies logically equivalent subcircuits within a netlist that are then handled separately during placement.

However, functional regularity alone does not imply placement will do a poor job. This has lead to the general industry practice of manually designing the datapath because of the possible significant timing and wire-length improvement possible. However, increasing design sizes and shortening turnaround-time demand a consolidated automated datapath extraction and placement framework. This work takes a new approach to extraction. Instead of searching for functional or structural regularity, this work extracts placement structures that are likely to be suboptimally placed.

To address this, Chapter 4 presents a high-performance placement flow with automatic datapath extraction and evaluation through high-dimensional data learning (PADE). The proposed flow is the first to handle modern large scale, mixed-size and mixed random logic/data path circuits. The key to this work is that the training data consists of circuits that are specifically known to be placed suboptimally within a design. This means the classification framework is not limited to datapath circuits alone but is extensible to any

structure known to be placed poorly by the placer.

Section 4.2 outlines the novel high-dimensional data learning, extraction, and evaluation algorithm for structured datapath extraction in PADE. It considers not only logic structures, but also placement hints from initial global placement results. Details of the extraction flow are presented in Section 4.3. Section 4.7 develops an optimal algorithm for datapath cell alignment selection (to be used for guiding data-path aware placement) using integer linear programming. The effectiveness of the PADE flow is demonstrated in 5.8 with significantly better results than previous state-of-the-art placement flows.

The third key discovery is that enhancing placement flows with structure constraints is not limited to wirelength improvements. In fact, significant power savings is possible through structured design constraints. Predicted for many years, power constraints have throttled performance scaling once experienced in multi-Ghz microprocessor design. These constraints now relegate process enhancements to minor speedups with little change expected in the near future. This necessitates costly power savings techniques such as multiple supply voltage islands, multiple threshold voltages, and aggressive power gating techniques. In spite of these efforts, power persists as the greatest challenge to both modern multi-GHz designs and low-power System on Chip (SoC)s in nanometer CMOS technologies. Complicating the issue is the increasing on-chip variation (OCV), which produces a significant drop in yield [27][62] resulting in stricter design guide rules. These effects are particularly poignant for clock design.

Compounding the power problem, skew requirements of multi-Ghz design has necessitated the need for a hybrid clock routing methodology where a low-skew global clock mesh overlays the entire die area followed by locally buffered clock trees [96] [94][98]. This methodology, often described as multi-source clock tree synthesis (MSCTS), still faces challenges from increases in process variation and tightening design and OCV constraints making it difficult to generate correct-by-construction LCTs. Additionally, within an individual design, there could be thousands of these LCTs making accurate design time modeling and optimization difficult because of the runtime impact. In practice, overly pessimistic constraints are often applied to minimize electrical violations on the LCT, which results in extra timing closure cycles, significant redesign, or even engineering change orders (ECO)'s.

This dissertation presents a novel approach to improving local clock tree power in Chapter 5 through structured local clock tree placement templates. Section 5.2 outlines the background and presents a motivating example displaying the significant capacitance reduction possible through structured latch cluster placement templates. Section 5.3 presents the proposed overall template development flow and Section 5.4 details a genetic latch placement algorithm for identifying optimized placement solutions. Structured template generation and redundancy removal is proposed in Section 5.5 and the decision tree classification with novel distance metric is described in Section 5.6. Section 5.7 illustrates how the templates integrate into a modern physical design flow and lastly, experimental results are presented in Section 5.8.

Chapter 2

Structured Datapath Placement: Is There Really a Problem?

There have been significant prior efforts to quantify performance of academic placement algorithms, primarily by creating artificial test cases that attempt to mimic real designs, such as the PEKO benchmark containing known optimas [8]. The idea was to create benchmarks with a known optimal solution and then measure how far existing placers were from the known optimal. Since the benchmarks do not necessarily correspond to properties of real VLSI netlists, the conclusions were met with some skepticism. This chapter presents two custom structed datapath designs that perform common logic functions with hand-designed layouts for each. The new generation of academic placers is then compared against them to see how the placers performed for these design styles. Experiments show that all academic placers have wirelengths significantly greater then the manual solution; solutions range from 1.75 to 4.88 times greater wirelengths. These testcases will be released publically to stimulate research into automatically solving structured datapath placement problems. This is an extension of the preliminary work presented in [92].

2.1 Motivation and Contributions

Automatic VLSI placement algorithms have improved significantly since the ISPD placement contests in 2005 and 2006 [51], [52]. These contests released 16 new placement benchmarks derived from industrial designs. The benchmarks contained a number of important features that were not present in the previous set of benchmarks: (i) they ranged in size from 211k cells to 2.18M cells, much larger than previous benchmarks (ii) they contained large fixed obstacles not seen in previous benchmarks (iii) they contained large movable objects which cover more than a single circuit row in height, a feature that was added to existing benchmarks [1].

In addition, the structure of the contest forced placement algorithms to optimize half-perimeter wirelength (HPWL), runtime and a target density, which is used in practice to improve both timing and routability of circuits in physical synthesis. Prior to the contests, few academic placers could solve these realistic problem instances, though that is certainly not true today [6] [12] [33] [66]. It is easy to observe that benchmarks are important to guide the development of practical placement algorithms.

Prior to these contests, there were attempts to quantify the suboptimality of placement heuristics. Hagen, et al. [24] had the idea of taking copies of small circuits and replicating them, then loosely connecting their ports together, in order to create a much larger benchmark. For example, by connecting four copies of a well-placed circuit together in 2 x 2 grid, they obtained a placement wirelength that was no more than four times that of

the original circuit. While interesting, this experiment is arguably unrealistic since these defined connections between the copies do not correspond to real logic functions. Furthermore, no pin locations are defined for the circuit (nor were there any for the original). This chapter overcomes both prior objections.

More recently, Chang, et al. [8] created the placement examples with known optima (PEKO) and placement examples with known upperbounds (PEKU) algorithms and released two sets of benchmarks with solutions that are known to be optimal or close to optimal. Optimality was achieved by adding nets to cells in configurations that cannot be shortened. In other words, they created a design where every net was a super-short net, though the pin distributions of cells matched that of a typical VLSI circuit. Reported results show wirelengths in the range of 1.43 to 2.40 times the optimal value. Again while interesting, these netlists did not correspond to any logic function at all. It could be argued that the PEKO and PEKU testcases are artificially hard and that no placer would ever need to solve them.

Given the renaissance in automated placement technology that has occurred, it seems like a good time to revisit this issue of quantifying placement algorithms. Perhaps this new generation is close to optimal, especially given that placer improvements are worthy of publication when they manage to obtain a 1-2% improvement in wirelength. However, unlike previous efforts, this chapter quantifies placement algorithms on useful logic function. It is accepted folklore that current placement tools do not perform particularly well on custom or structured designs. Due to their regular structure, datapath designs

enable a designer to construct highly compact custom layouts. To shed light on this issue, the solutions of academic placement tools are compared on two manual designs created for this purpose.

The initial design for each circuit was developed using standard, custom design practices. Logic gates from automated design standard cell libraries were hierarchically built within a custom schematic design framework. Each design used a reduced library of basic 2-, 3-, and 4-input NAND, NOR, INVERT, MUX and XOR gates and latches. These gates were manually placed and fixed for both benchmarks and design inputs and outputs placed directly on the driving or receiving gate. The combinational logic gates for both benchmarks were allowed to move during the course of automatic placement by several academic tools [77]. It is observed that every placer produced a solution having wirelength at least 1.75 times that of the custom solution while having no density constraints.

Many have speculated that poor performance of placers on datapath designs is due to very tight density constraints. Perhaps placers could find the right structures, but simply had trouble with the legalization. Consequently, eight variants of each design were created where additional whitespace was inserted to provide more opportunity for the placers. While wirelength was improved, all placers still generated solutions with wirelengths at least 1.44 times that of the custom solution. The empirical results confirm there remains significant room for improvement in modern academic placement algorithms.

The chapter is organized as follows. Section 2.2 presents a rotate circuit,

and shows a common manual layout solution. Section 2.3 does the same for a compare logic circuit. Experiments results comparing six placers are presented in Section 2.4. The next section analyses the experimental results.

2.2 Rotate Logic

Rotate circuits, also known as cyclic shifters [25] [39] [17], are a simple and common bit operation generally found throughout microprocessors, cryptography, imaging, and biometrics [20] [50]. Traditionally, rotators are custom designed because of their highly regular structure and significant routing complexity [19] [75] [18] though some work on automated placement has been explored [26].

2.2.1 Rotator Overview

A standard rotate function consists of cascaded 2-input MUXes, as shown in Figure 2.1. A rotator circuit receives a set of inputs $d[0 : n-1]$ and $r[0 : m-1]$ and produces an output $s[0 : n-1]$, where $d[0 : n-1]$ has been rotated by some amount encoded by $r[0 : m-1]$. In the following notation, & indicates a logical AND, + indicates a logical OR, and ! indicates a logical NOT. To mathematically define the rotate functions, let $k[i, j]$ denote the internal point at ith row and jth column in 2.1, where $i = (0 : m-1)$ for $r[0]$ to $r[m-1]$ and $j = (0 : n-1)$ for $d[0]$ to $d[n-1]$. Then, $k[0, j] =!(r[0])$ & $d[i] + r[0]$ & $d[i+1]$, where $j = 0, ..., n-1$ and note that $n = 2m$.Thus the general equations are:

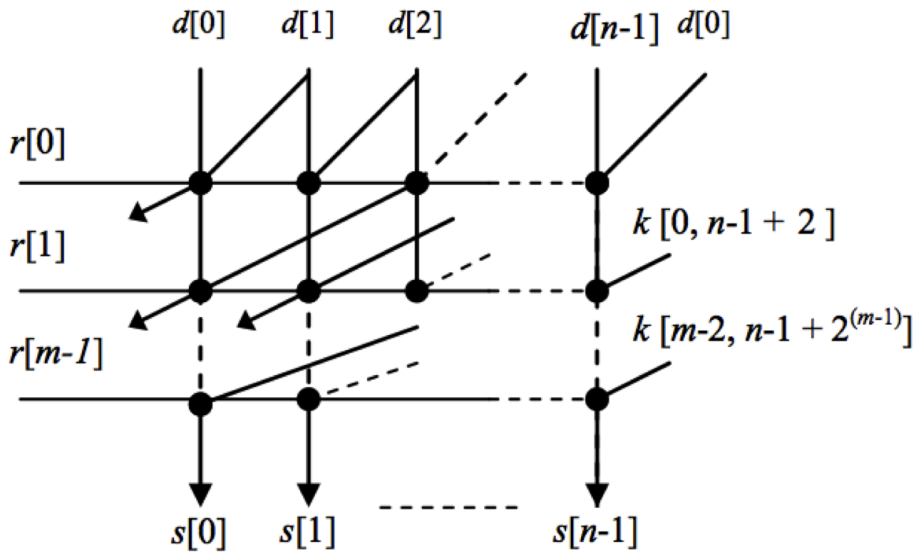


Figure 2.1: A rotate block diagram receiving the set of inputs $d[0:n-1]$ and $r[0:m-1]$ and producing the output signal $s[0:n-1]$, where $d[0:n-1]$ has been rotated by some amount encoded by $r[0:m-1]$

.

$$\begin{aligned} k[i,j] =& !(r[i])\ \&\ k[i-1,j] + r[i]\ \&\ k[i-1,j+2i] \\ & \text{where } i = 0,...,m-1,\ \ j = 1,...,n-1 \\ & k[i,j] = k[i,j+z*n], \\ & \text{where } z = 0,1,2,... \end{aligned} \tag{2.1}$$

Figure 2.2 shows an example of an eight-way rotate function. The initial input vector $d[0:7] = 01110101$ and $r[0:2] = 101$, indicating a rotation of five. In the first stage, $r[0]$ rotates the input vector one bit position, $r[1]$ in stage two does not rotate the vector, and in the third stage, $r[2]$ rotates the vector four more bit positions for a result of $s[0:7] = 10101110$.

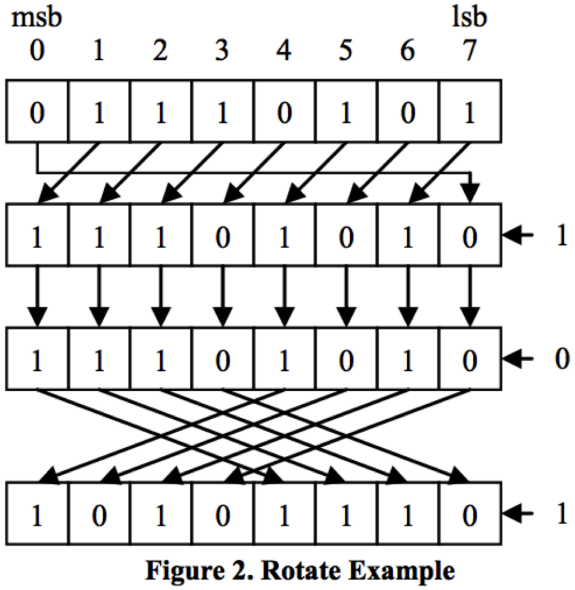


Figure 2.2: An eight-way rotate logic function example. The initial input vector $d[0:7] = 01110101$ and $r[0:2] = 101$, indicating a rotation of five.

Rotator designs present automated design tools with the challenge of producing a densely-packed placement solution while minimizing routing congestion. There are two parts to the routing challenge, local routing and global routing. Local routes between each MUX must be lined up very carefully to leave space for the global select lines $r[0:m-1]$. At each stage, the route from the previous stage shifts one more column over, creating a congested routing network. Design placement that minimizes jogging global routes is critical to achieve a routable design that meets area and timing constraints. In addition, careful attention to the design of global routes is necessary for optimal delay.

2.2.2 Rotator Benchmark Details

The first benchmark in this paper, Design 1, derives from the manual placement of an actual high-speed microprocessor rotate function. The logic implementation also includes two enable signals at each rotate circuit. Certain portions of the design are modified, such as the intermediate output pins and the latch points, without modifying overall functionality. Figure 2.3 displays the basic MUX and the enable building block for the design, which is referred to as a complex subcell. Each complex subcell is comprised of a two-to-one MUX with a corresponding select signal r[i] and enable signals $e_h[i]$ and $e_v[j]$. Enable signal $e_h[i]$ runs horizontally to each bit stack in the ith row, and $e_v[j]$ runs vertically to each complex subcell within a bit stack at jth column. Exact circuit implementation can vary, depending on the specific technology; however, in this design, a single two-to-one MUX and a three-input NAND gate are used for the implementation. Each following stage is inverted to maintain polarity without impacting TWL calculations.

Using the notation from Figure 2.2, Design 1 contains $n = 511$ and $m = 63$, which means it is a 512 bit rotate circuit with 9 encoding bits. Each $d[0 : n - 1]$ is stored in a latch with a fixed location and drives the stacked MUX structure, which is nine complex subcells high. Primary input (PIs) and output pins (POs) were placed directly on top of their respective connections minimizing PI/PO routing distance.

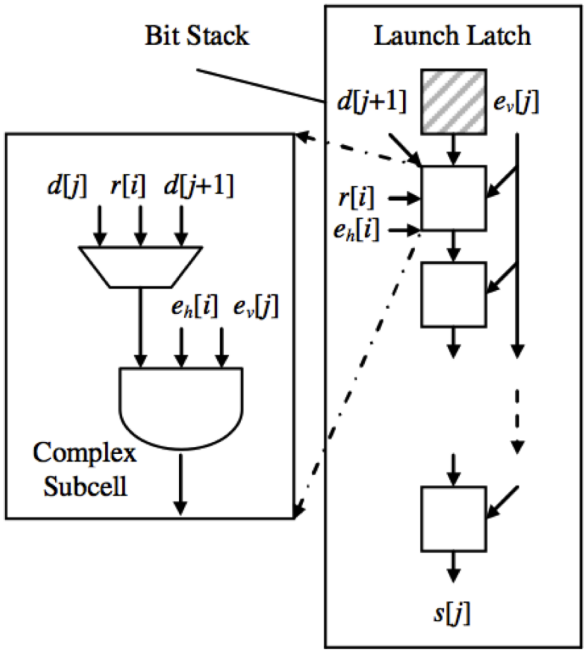


Figure 2.3: Rotate sub block consisting of a basic MUX and enable gate that is referred to as a complex subcell. Each complex subcell is comprised of the two-to-one MUX with a corresponding select signal r[i] and enable signals $e_h[i]$ and $e_v[j]$.

2.2.3 Placement Details

Figure 2.4 shows a representative layout for an 8-bit rotator as in Figure 2.2 . The data bus $d[0:7]$ initially resides in the latches denoted lat with each complex subcell stacked directly on top. The mux is the 2:1 MUX in the complex subcell as is the AND-3 gate in the complex subcell. The rotate result, $s[0:7]$ leaves the top of each bit stack driven from the last complex subcell.

Figure 2.5 displays the next level of hierarchy in which bit slices are

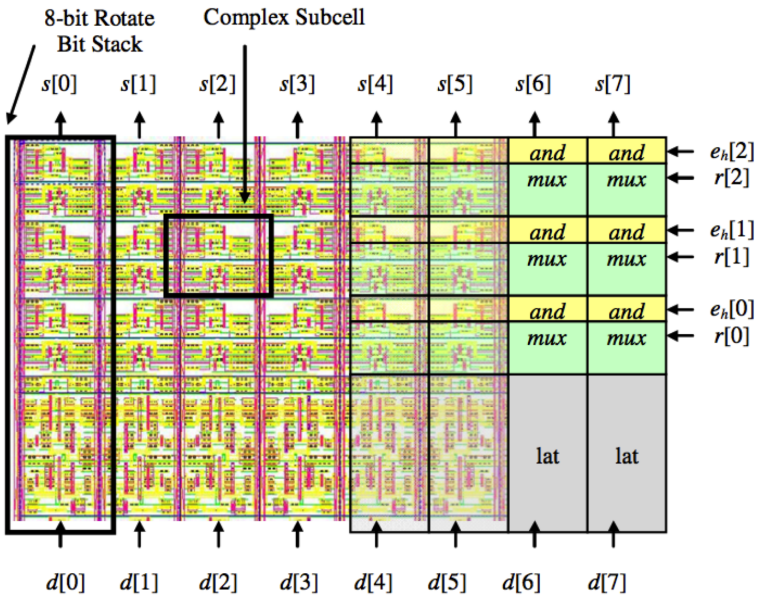


Figure 2.4: The representative layout for an 8-Bit rotator with 3-bit encoding.

placed next to each other to form a rotate row, n-bits wide. Let α denote the total latch height, β denote the total logic height of the stacked complex subcells (nine in this example, corresponding to $r[0:8]$), and ϵ denote the any added whitespace in the bit slice. Each bit stack is ordered, as in Figure 2.3, to line up the MUX rotate signals $r[i]$ and enable signals $e_h[i]$ and $e_v[j]$ with their corresponding complex subcell. This is critical for both routability and minimizing TWL, since the fanout on $r[i]$, $e_h[i]$ and $e_v[j]$ is very large. Between bit stacks $(n/21)$ and $(n/2)$, space for buffer placement is added where the rotate line bus $r[0:m-1]$ and enable signal bus $e_h[0:m-1]$ are lined up to drive horizontally to each bit slice.

The top level of hierarchy is shown in Figure 2.6 where each row from

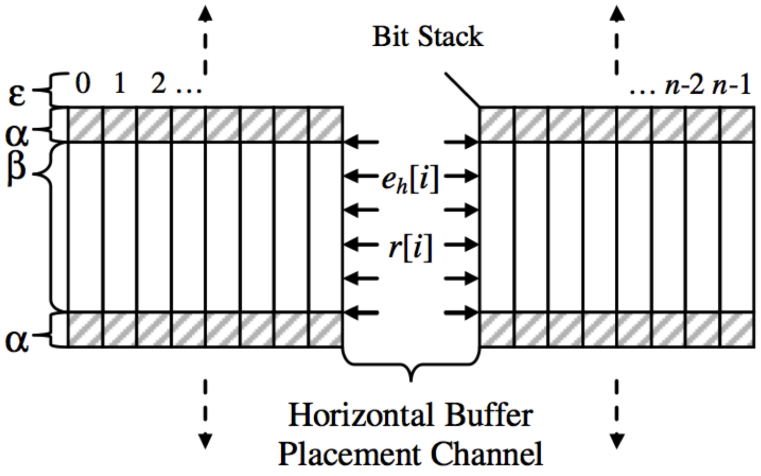


Figure 2.5: Rotator row where bit slices are placed next to each other, n-bits wide.

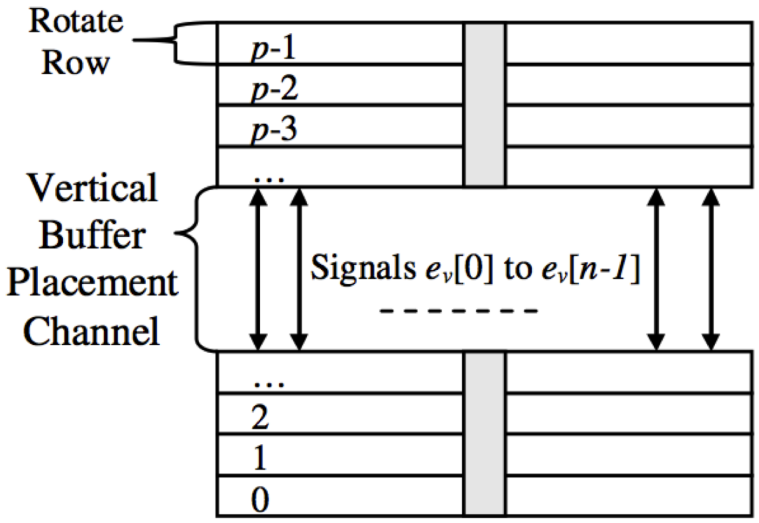


Figure 2.6: The top level of hierarchy of the rotator placement where each row from 0 to $p-1$ is an independent copy of the rotate row shown in Figure 2.5.

0 to $p-1$ is an independent copy of the rotate row shown in Figure 2.5. In the middle of the block, space for buffer placement is added where the enable

signal bus $e_v[j]$ is lined up to drive vertically to each row.

2.3 Enable Logic

Design 2 is a standard enable AND/OR logic tree for [25] [39] common throughout datapath design [1] with the bit stack logic structure shown in Figure 2.7. This structure is used in many applications, such as translation buffers and structured content addressable-memory circuits. Two signals, a_o and $s_d[j]$, are driven into an AND gate with the data inputs and then into an OR tree. The output of the OR tree is then ORed with a set signal e_i, and the result is latched.

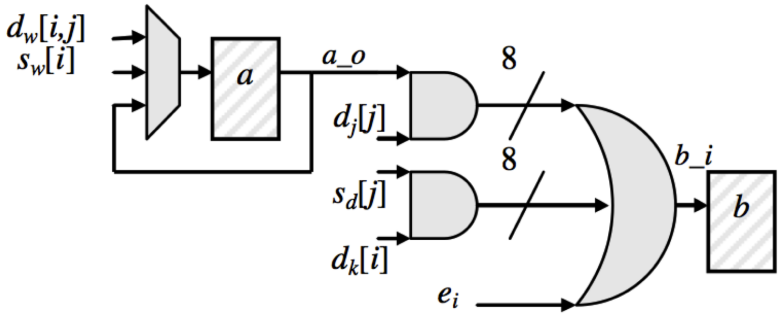


Figure 2.7: Design 2 bit stack where two signals, a_o and $s_d[j]$, are driven into an AND gate with the data inputs and then into an OR tree. The output of the OR tree is then ORed with a set signal e_i, and the result is latched

[1] Standard cell design practice allows for the interchange between logic gates of the same size. Thus, this implementation can be modified to many representative circuits, such as magnitude comparators, standard equality circuits or parity circuits.

2.3.1 Benchmark Details

Design 2 is a simplified version of a custom placed industrial design. Careful packing of the repeated logic enables optimization of both timing and area while reducing congestion. The bit stack in Figure 2.7 is repeated $n = 257$ times in one row and there are $m = 32$ rows placed within Design 2. Signal $d_w[i, j]$, $d_j[i]$, and $d_k[i]$, where $i = 0, 1, ..., n-1$ columns and $j = 0, 1, ..., mrows$, are primary data input signals; e_i and $s_d[i]$ are high fanout select lines running through the ith row where $i = 0, 1, ..., n - 1$, and $j = 0, 1, ..., m - 1$ for the bit stack with m rows and n columns. Select line $s_w[i]$ runs within row i and is a write enable select signal to latch new data into latch a. If $s_w[i]$ is not enabled, the prior value in a is selected and stored. Enable signal e_i is an override signal that will set latch b.

2.3.2 Placement Details

Custom placement of Design 2 leads to regularly placed rows with tightly packed cells, shown in Figure 2.8, where eight total bit stack cells have been placed. Figure 2.8 represents a partial 8 bit stack lay out for illustrative purposes of the manual layout solution. In the full implementation, each bit stack consists of 16 AND gates driving a 16 way OR gate configuration. The logic gates from Figure 2.7 are interleaved into a single circuit row, and pins between rows for each select line are lined up evenly to reduce branch routing. Latch a and the MUX that drives it are placed at the bottom of the stack, the data flows through the AND/OR reduce logic.

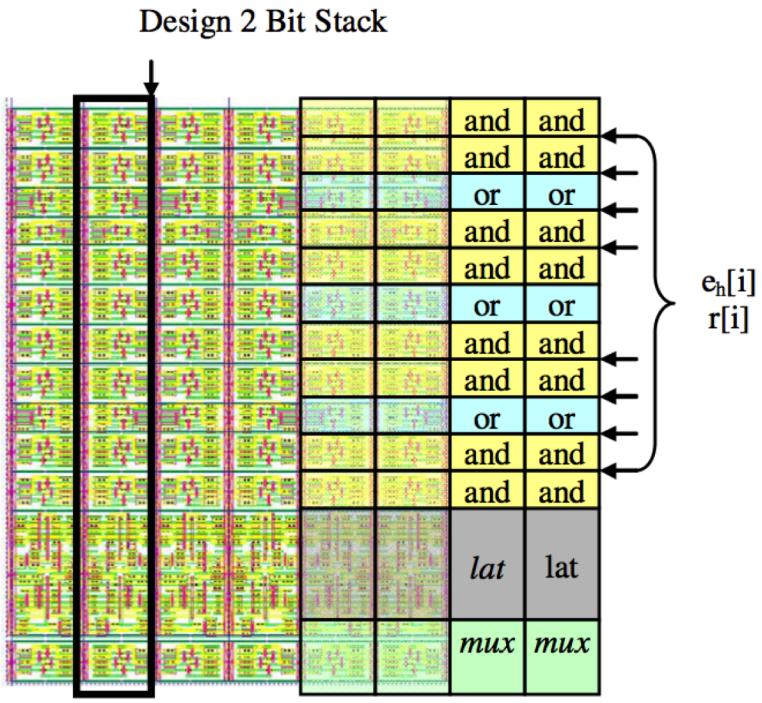


Figure 2.8: Design 2 physical layout of an 8 wide Design 2 bit stack.

Figure 2.9 shows the overall layout of the entire design with one bit stack shaded. Each bit stack is placed side-by-side n=257 times in one row, where there are 12 rows in total and placement space is added in the middle, both vertically and horizontally for the global wire drivers.

2.4 Experimental Results

Table 2.1 outlines the design details for each circuit that was constructed. Design 1 contains 140,800 movable cells, and Design 2 contains 130,944 movable cells. These are both reasonably small, custom-placement

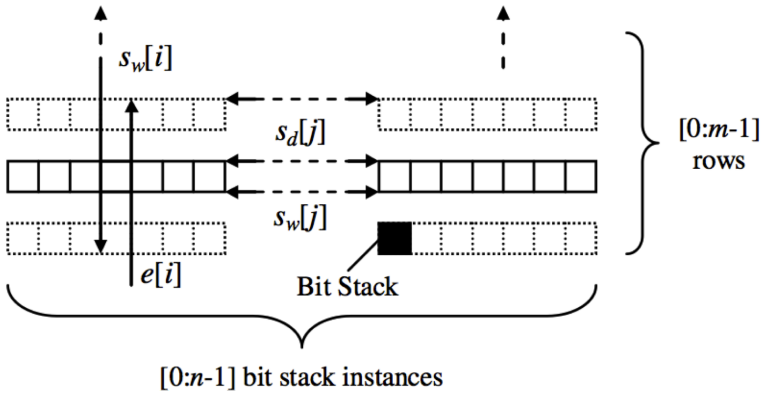


Figure 2.9: overall physical layout of resign 2 with one bit stack shaded. Each bit stack is placed side-by-side n=257 times in one row, where there are 12 rows in total and placement space is added in the middle

designs built using common structured placement tools, including schematic capture and layout. Once built, the netlist was exported to Bookshelf [51] [52] format and the wirelength measured. The custom layout solution is compared against the following placers for these two designs:

1. mPL6 v6 [6]
2. CAPO v10.2 [66]
3. FastPlace v3.0 [83]
4. NTUPlace3 v7.10.19 [12]
5. APlace v1.0 [33]
6. Dragon v3.01 [87]

Table 2.1: Benchmark Design Characteristics

Design	Net Count	Pin Count	Terminal Count	Movable Cells
Design 1	157849	637984	19488	140800
Design 2	148682	661340	21724	130944

The authors of Timberwolf [77] were contacted, but they were unable to provide a version compatible with the Bookshelf [51] [52] format at this time. This simulated annealing approach may perform well on these moderately sized test cases. For all placers, a target density constraint2 was not imposed to give maximum freedom to pack cells. [2]

2.4.1 Initial Placement Results

Table 2.2 displays the results of the custom placement versus the academic placers. Column one shows all different placement algorithms, where "custom" corresponds to the manually placed designs. Column two displays the measured TWL compared to the custom solution for each placement method on Design 1, and column five displays the TWL for Design 2. For both designs, APlace failed to find a legal placement solution. Columns three and six correspond to the percentage increases in TWL compared to the custom-placed solution. Columns four and seven display the placement runtime in seconds for each design. The custom-placement method resulted in a TWL of 11,000,365 for Design 1 and of 8,642,097 for Design 2. For Design 1, all placers completed with overlaps. Of the placers that completed, CAPO produced the

[2] This was achieved by supplying each placer with a target density requirement of 100% density as defined as in ISPD placement contests [51] [52]

best automated placement result with 15945589, a 45% increase in TWL. For Design 2, the ntuPlace3 algorithm resulted in the best automated placement result with 10,765,222, a 1.25 TWL ratio. The run time for all placers for both designs is very fast because of the small design size.

Table 2.2: TWL results on fully highest density design. Starred (*) items completed with overlaps and n/a entries did not complete for the base case

	Design 1			Design 2		
Placer	TWL	TWL Ratio	Time (s)	TWL	TWL Ratio	Time (s)
Custom	11000365	1.00	n/a	8642097	1.00	n/a
CAPO	15945589*	1.45	n/a	n/a	n/a	n/a
mPL6	18290965*	1.66	n/a	n/a	n/a	n/a
ntuPlace3	n/a	n/a	n/a	10765222	1.25	533.0
APlace	n/a	n/a	n/a	n/a	n/a	n/a
Dragon	52926316*	4.81	2350.18	34711167	4.02	2692.0
FastPlace	16336840*			n/a	n/a	n/a

2.4.2 Adding Additional Whitespace

As mentioned earlier, it is important to understand whether placers could not find the right structure, or could not legalize. Seven additional variations of each benchmark were generated by increasing white space using the following scheme. As shown in Figure 2.10, let η denote the total height of default bit slack, α denote the height of latch logic, β denote the height of placeable logic, and ϵ denote the white space added to the bit stack.

The original testcase experiment for both designs was set up with no extra white space between each row. Then the white space was increased using the scheme in Figure 2.10 and manually replaced the custom solution so that

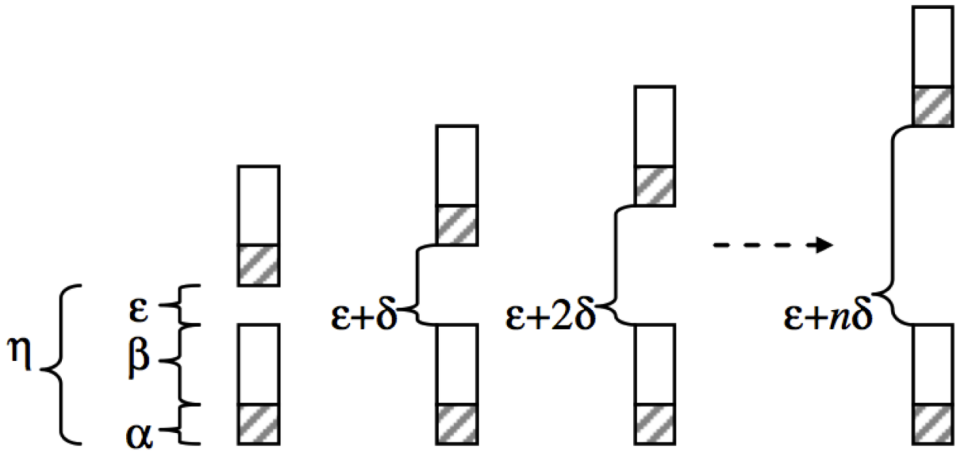


Figure 2.10: Structured Placement Experiments

the automated placement algorithms can be compared.

2.4.3 Whitespace Placement Results

Eight experiments were run on both designs, incrementally increasing the available whitespace to allow more room for automated placement tools while not applying any constraints. Tables 2.3 and 2.4 display the TWL placement results for each experiment compared to the custom design solution at the same whitespace percentage [3].

All placers except Dragon show wirelength improvement as more whitespace is added with a slight increase with the most whitespace. When more than 15% of whitespace is added however, the improvement of TWL and its ratio starts to saturate. After that point, adding additional white space did not

[3] ntuPlace3 did not complete for Design 1 and APlace failed to legalize for both designs.

Table 2.3: TWL Results for Design 1

Placer	92.5	89.0	85.8	82.8	80.1	77.4	74.0	71.9
Capo	1.45	1.49	1.24	1.28	1.14	1.18	1.12	1.11
mPL6	1.66	1.65	1.64	1.66	1.64	1.66	1.76	1.73
ntuPlace3	-	-	-	-	-	-	-	-
APlace*	-	-	-	-	-	-	-	-
Dragon	4.81	5.00	5.39	5.88	5.83	5.91	6.56	7.37
FastPlace	1.47	1.33	1.31	1.31	1.28	1.26	1.28	1.31

Table 2.4: TWL Results for Design 1

Placer	96.1	93.6	89.5	85.3	81.5	78.1	75.2	72.2
Capo	1.66	1.24	1.17	1.18	1.18	1.20	1.20	1.21
mPL6	-	1.19	1.15	1.72	1.15	1.16	1.17	1.18
ntuPlace3	1.29	1.12	1.14	1.13	1.20	1.15	1.16	1.24
APlace*	-	-	-	-	-	-	-	-
Dragon	4.02	4.24	4.49	4.81	5.09	5.33	5.60	5.93
FastPlace	-	1.26	1.15	1.15	1.17	1.19	1.20	1.21

significantly improve the overall TWL ratio compared to the custom placed design. Results for APlace are not shown because it failed to find a legal placement solution. One interesting result is the significant TWL increase in Dragon placer as available whitespace is added. In general, the overall TWL ratios are improving as whitespace increases however; this is primarily due to the TWL increase of the custom solution.

For Design 2, TWL at only 4% whitespace is significantly higher for all placers but quickly drops. The custom TWL for Design 2 increases significantly as whitespace increases for the design helping the overall TWL ratios for the placers. This again does not point to an improved placement solution with increased whitespace, but is instead a result of the logic spreading from

the manual solution. Dragon placement results also exhibit the significant TWL increase trend seen in Design 1 as whitespace increases. Minimum overall TWL for the placers occurs within the range for 7% to 20% whitespace after which TWL begins to increase at a similar slope to the custom solution.

The following observations were made:

1. Generally, placers did not improve much with additional white space, with the exception of CAPO, which improved from 1.66 to 1.17 on Design 2. Part of the improvement comes from the TWL increase from the manual solution.

2. For Design 1, the best overall result for each experiment came from CAPO with a 14% increase in TWL at 80.1% utilization. The TWL increase of the manual solution is not obvious, but it also increases with added area.

3. There was a gradual improvement in the TWL ratio for Design 2 as area increased for the design, with ntuPlace3 decreasing from 1.29 to 1.12.

4. By industry standards, both designs are small relative to state-of-the-art work yet all placers presented significantly suboptimal TWL results.

5. All placers failed to place Design 1 without overlaps. Though some of the placers completed Design 2 without overlaps, a 15% degradation in TWL translates to significant power and delay increases compared to a custom solution.

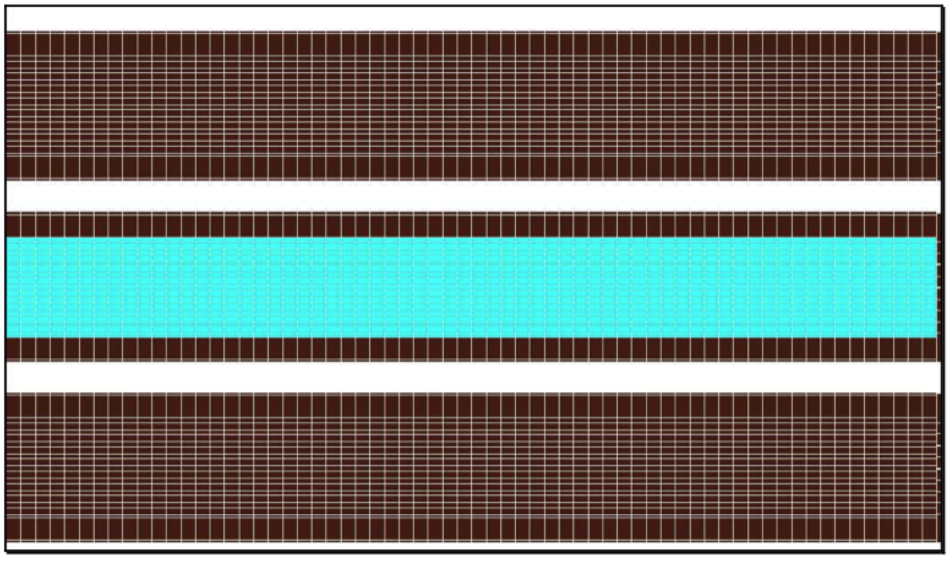


Figure 2.11: Custom Design 1 Placement Solution

2.5 Analysis

Obviously, it is disappointing that academic placers perform poorly on these real designs, so further examination is necessary. Figure 2.11 displays a zoomed in snapshot of three rows of the custom placed layout for Design 1 with the placeable logic between the latches from one row highlighted in light blue. The design was placed again using one of the better performing placement algorithms to see what happens to the blue cells that are densely packed in the custom layout. This is shown in Figure 2.12.

Observe the irregularity present in the blue highlighted logic. Most of the blue logic is placed within the bounds of the correct rows of latches, but a significant portion is left outside, despite adequate whitespace. This may occur because:

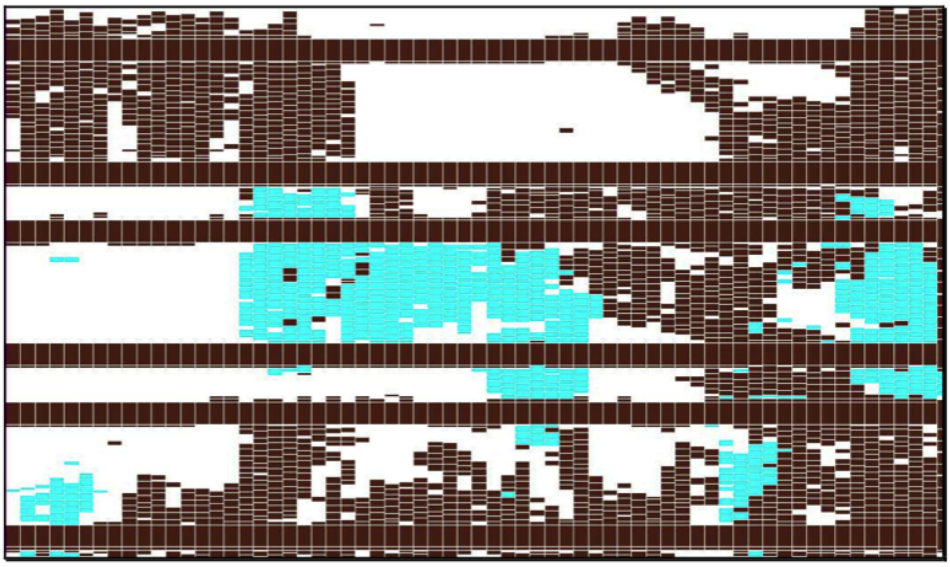


Figure 2.12: Automated Design 1 Placement Solution

1. Multilevel placement algorithms employ clustering to abstract a netlist into larger components that will be placed together. This manifests itself in loosely connected "blobs" of logic rather than densely packed structures.

2. Analytical placement algorithms commonly employ net models such as cliques or stars of two-pin edges to represent hyperedges. Such models derate the weights of edges representing high-fanout nets to compensate for the increased number of edges needed to represent them. As a result, the impact of a 512-bit select line is very low, yet these nets could provide clues to the structure within the design.

3. Clustering algorithms do not typically account for logic functions, and

make decisions purely on local connectivity. This often leads to merging of gates across bit slices rather than merging the slice into a large cluster.

2.6 Summary

Recent years have seen truly significant improvements in runtimes, quality, and scalability in standard-cell placement algorithms. This chapter measures their performance on real datapath placement examples and compares them to hand-designed layouts. Academic placers still have a long way to go in order to match the quality of custom design solutions. An important contribution of this chapter is to release these benchmarks publically.

In order to keep pace with technology innovation, design automation must strive to improve productivity. This chapter highlights poor performance of modern placement tools on a key design style that is lacking in automation, namely structured datapaths. Results of this chapter lead to discoveries in three key aspects of modern physical design flows, the next being exactly why placers perform poorly on these designs.

Chapter 3

Structure-Aware Placement Techniques for Designs with Embedded Datapaths

A key discovery identified by the structured placement benchmarks is that analytical placement of high fanout nets disrupt structured circuits causing significant routed wirelength degradation. Additionally, these structures are often deeply embedded within ASIC style logic circuits increasing the complexity for minimizing global wirelength. Thus, a single unified placement flow handling both structured and random logic simultaneously is extremely valuable, improving design time, solution quality, and saving development and maintenance costs.

This chapter shows that with careful design guidance on high fanout nets, existing Half-Perimeter Wirelength (HPWL) driven placers can better handle designs with embedded structured logic. It is accomplished via a set of effective placement techniques amenable to incorporation within existing random-logic placers. The flow leverages the speed and flexibility of state-of-the-art HPWL-driven placers, while imposing alignment constraints[1] to

[1]Alignment constraint was also discussed in [34] as an example of geometric constraint handling, but no circuit structures were considered.

achieve better regularity and Steiner Wirelength (StWL). This is an extension of the preliminary work presented in [90].

3.1 Introduction

As ASIC frequencies exceed $1GHz$ and shrinking schedules drive increased automation for microprocessor designs, the boundary between manually designed datapath-logic and random-logic macros is blurring. A new *hybrid* design style is emerging, wherein designs contain both random logic and datapath logic. The datapath logic generally refers to circuit structures containing highly parallel bit operations [54] (often called the bit-stack), and careful design is important for high frequency designs. Prior work [31] has shown that handling the datapath-logic placement independent of the random logic, overly constrains the random-logic placement, degrading overall congestion and wirelength. A single placement flow handling both datapath and random logic is extremely valuable, improving design time, solution quality, and saving development and maintenance costs. However, [92, 61] demonstrate that most state-of-the-art placers are incapable of handling designs with regular structure. This chapter shows that with design guidance, existing HPWL-driven placers can better handle designs with embedded datapath logic.

This chapter presents a novel structure-aware placement flow for hybrid designs via a set of effective placement techniques amenable to incorporation within existing random-logic placers. The flow leverages the speed and flexibility of state-of-the-art HPWL-driven placers, while imposing alignment

constraints[2] to achieve better regularity and StWL. The key contributions of this chapter are as follows:

1. A study of the issues with current academic placers: the inadequacies and specifically the lack of fidelity of the HPWL model versus the StWL model when evaluating and placing datapath logic.

2. A key insight to bit-stack alignment: alignment of the bit-stack guides indirect StWL optimization, and significantly improves total StWL and routing congestion.

3. A novel placement flow: Structure-Aware Placement Techniques (SAPT) that can be incorporated within existing HPWL-driven placers to enable better alignment of the embedded datapaths during both global and detailed placement.

Section 3.2 outlines the problem faced by current random-logic placers when placing datapath logic. Section 3.3 presents the preliminaries and placement definitions. Section 3.4 provides an overview of the structure-aware placement flow with general descriptions of each technique. The structure-aware global placement techniques are described in Section 3.5 and structure-aware detailed placement techniques are described in Section 3.6. Section 3.7 presents experimental results, followed by a summary in Section 3.8.

[2]Alignment constraint was also discussed in [34] as an example of geometric constraint handling, but no circuit structures were considered.

3.2 Motivation and Background

A common assumption among IC designers is that circuits with high regularity such as datapath logic require manual placement. Perpetuating this assumption are two key factors that limited adoption of past automation attempts. First, prior approaches separate control logic placement from datapath-logic placement. Second, a prevailing evaluation metric for random-logic placement, HPWL is inadequate for structured circuit styles. This section addresses each of these factors, by first establishing the need for a unified placement framework and then highlighting the inadequacy of the HPWL metric for regular structures. It then demonstrates that cell alignment during placement implicitly optimizes StWL, producing significant wirelength improvements for datapath style circuits.

3.2.1 The need for a unified placement framework

Automatic placement of structured circuits has been performed by dedicated datapath placers such as [71, 100, 54], which generate highly compact, area efficient placements. After layout generation, these methods construct a larger macro block or small individual bit-slice macro blocks, followed by the main random-logic mixed-size placer. More recently, promising results were presented in [14] where an innovative row-based placement of the datapath is proposed instead of the traditional bit-stack alignment. A nonlinear optimization for HPWL minimization with a sigmoid-based density model for density control in datapath circuits is proposed. Once datapath placement completes,

the cells become a movable macro. Generalizing this approach, results show it is possible to separately place the datapath cells and then apply mixed-size placement techniques to generate significantly smaller HPWL than prior placers.

However, these techniques suffer from some key drawbacks: First, even though a datapath placer may minimize the local wirelength through cell ordering [23], or optimizing specific bit-stacks [99], global interconnect optimization with the embedded datapath is not taken into account simultaneously during placement. Second, by making the datapath a macro, in the general case, the datapath macro layout must occur first and it is very difficult to select and optimize the correct macro aspect ratio. Third, in industrial hybrid designs, the datapath is not always tightly packed but many cases still require alignment. Fourth, packing may force other more critical random logic out of a specific area, reducing overall quality of results (QOR).

3.2.2 StWL and HPWL comparisons for datapath circuits

Datapath circuits are typically driven by one or more high-fanout nets. Traditional HPWL-driven placers naturally compact the placement of high-fanout nets to minimize total wirelength. However, known optimal layouts for many regular datapath structures are drastically different [40], often not corresponding to a minimum-HPWL placement solution. To illustrate this point, Table 3.1 compares a few state-of-the-art academic placers using both, total HPWL and total StWL on the modified ISPD 2011 Datapath Bench-

Table 3.1: Legalized HPWL and StWL comparison on the ISPD 2011 Datapath Benchmark A [91] between manually placed and automated placement solutions. Placement results are sorted by increasing HPWL value. To note: (1) Best HPWL solution does not indicate the best StWL solution. (2) Bold numbers are the best automated placement wirelength.

	ISPD Datapath Benchmark A			
	Total HPWL		Total StWL	
Manually Placed	11000365	1.00	11066683	1.00
CAPO v10.2 [68]	11535525	**1.05**	21516128	1.94
SimPL [37]	11837307	1.08	20180311	**1.82**
mPL6 v6 [6]	12919955	1.17	23950663	2.16
NTUPlace3 v7.10.19 [12]	13447753	1.22	24673151	2.23
FastPlace v3.0 [80]	15672727	1.42	27115750	2.45
Dragon v3.01 [87]	16424739	1.49	26182449	2.37
	ISPD Datapath Benchmark B			
	Total HPWL		Total StWL	
Manually Placed	8642097	1.00	9823680	1.00
CAPO v10.2	10338805	**1.20**	23881606	2.43
NTUPlace3 v7.10.19	10433894	1.21	26110039	2.66
SimPL	10631304	1.23	22319594	**2.27**
Dragon v3.01	12229019	1.42	28577316	2.91
FastPlace v3.0	14537026	1.68	36642434	3.73
mPL6 v6	16263018	1.88	28846387	2.94

mark Suite [91] [3]. All StWL measurements were performed using coalesCgrip [74], and all reported numbers are total wirelength results for each design. The HPWL column in Table 3.1 is sorted from smallest to largest for each benchmark. In addition, the table reports the wirelength ratio normalized to the manually placed solution. Careful examination of this table yields the

[3]The MISPD 2011 Datapath Benchmark Suite was modified to contain unfixed latch rows compared to the original fixed latch placement reported in ISPD 2011. Benchmarks can be downloaded at: http://www.cerc.utexas.edu/utda/download/DP/[92]

following surprising results:

1. While HPWL from the automated placement solutions for both benchmarks are very close to the manually placed solution, the StWL results degrade significantly, with the best automated solution at 1.82× in StWL for benchmark A and 2.27× for benchmark B compared to the manual solution.

2. Fidelity of the HPWL metric appears low for datapath logic. As shown in Table 3.1, the HPWL column is sorted by increasing value and it is generally expected that StWL would maintain the same order, but in fact that does not happen. Additionally, for both benchmarks, the placer with the best HPWL (Capo) is not the placer with the best StWL (SimPL).

As shown in Section 3.7.4, the significant improvement in StWL also corresponds to vastly improved congestion metrics. There has been prior work in directly optimizing StWL [67] with the Rooster placer. As reported in [67], StWL has much better correlation to the routed wirelength (rWL) as compared to HPWL. However, as results show in Tables 3.7 and 3.8 for Rooster, optimizing StWL alone does not effectively address the alignment requirements of datapath circuits. Additionally, HPWL is easy to compute, and is a reasonable first-order estimate of timing and power on vast number of random-logic designs. This makes it a popular objective to optimize during placement.

Therefore, instead of completely changing the placement objective, this chapter presents techniques to improve the placement quality of datapath logic while retaining existing HPWL-driven placement frameworks.

3.2.3 Implicit StWL optimization through bit-stack alignment

In Figure 3.1(a), a partial logic netlist with one NAND gate, shown as hashed, drives net *net*1 with a fanout of 10. All the input and output pins are fixed objects placed on top of the gate. Figure 3.1(b) shows a manually placed solution for this partial circuit and Figure 3.1(c) shows a solution from an existing placer. The dark-shaded cells match the same dark-shaded NAND gates in Figure 3.1(a). The light-shaded gray cells represent other logic placed within the design.

For both solutions, the total HPWL and StWL numbers are shown in Figure 3.1. As indicated in Section 3.2.2, even though the HPWL of the manual solution (1442) is greater than the HPWL of the automated placement (1415), the StWL shows the reverse trend. While it is impractical to list HPWL and StWL of every single net, clearly for net *net*1, the StWL in Figure 3.1(b) is better than the StWL in Figure 3.1(c). This is due to the better alignment of the structured cells in one horizontal row, which produces much better StWL. Also the solution of Figure 3.1(c) shows the existing placer compacting the placement of the net in both x- and y-directions to lower HPWL, but degrading StWL. This example shows that if a HPWL-driven placer can obtain better alignment for regular structures, it can implicitly have better StWL, without

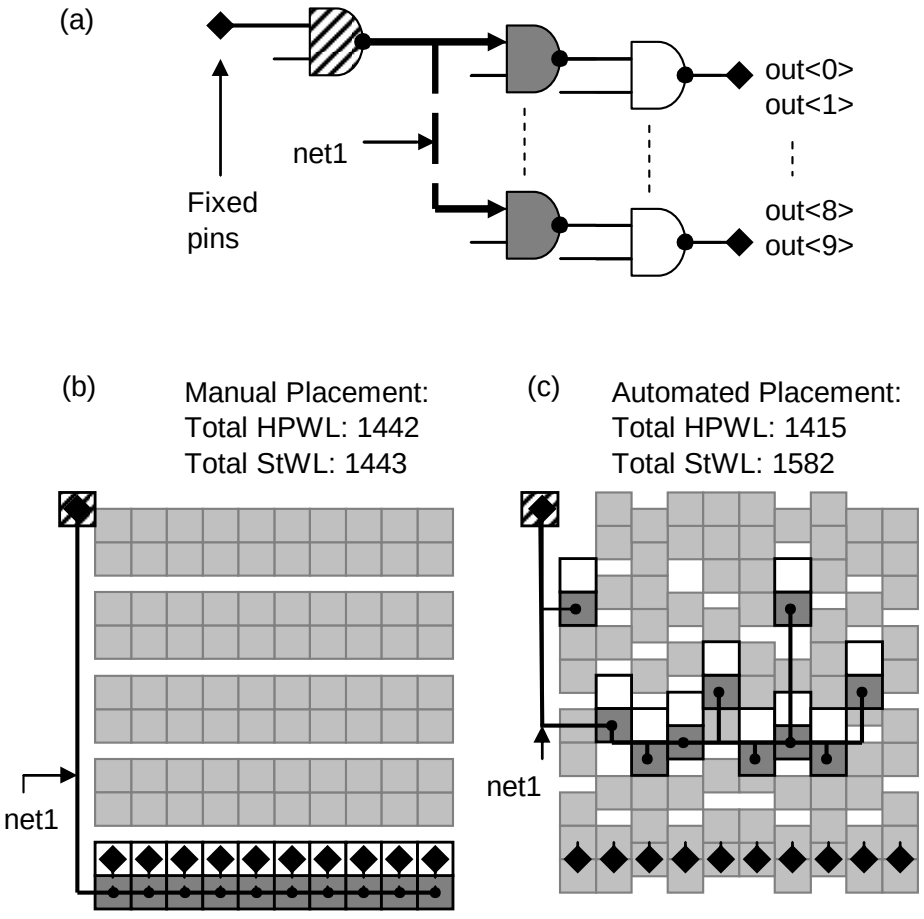


Figure 3.1: An example circuit where StWL of the manually placed design is better than that of the automated placement, but HPWL of the automated placement solution is better than that of the manual placement. Net1 has fanout of 10.

having to optimize for it directly.

Motivated by the above examples, new techniques are developed to guide an existing HPWL-driven random-logic placer to generate a placement similar to Figure 3.1(b), with better StWL than the one in Figure 3.1(c). Addi-

tionally, by providing alignment constraints to small portions of the datapath, it is observed that, during the iterative placement process, other surrounding cells become aligned as well. This can be observed visually in the placement results in Figure 3.8 where only some of the cells have been manually defined.

The alignment constraints presented in this chapter provide hints to placers, directing them toward more globally-optimized solution. As results will show, with relatively few manually defined bit-stacks, the framework significantly reduces overall wirelength and congestion.

3.3 Preliminaries

Given a netlist $N = (V, E)$ with nodes V and nets E, placement obtains locations (x_i, y_i) for all the movable nodes, such that the area of nodes within any rectangular region does not exceed the area of cell sites in that region.

With $\vec{x}, \vec{y} = \{x_i, y_i\}$, the HPWL is defined as:

$$HPWL(\vec{x}, \vec{y}) = HPWL(\vec{x}) + HPWL(\vec{y}) \tag{3.1}$$

$$HPWL(\vec{x}) = \sum_{e \in E} [MAX_{i \in e} x_i - MIN_{i \in e} x_i] \tag{3.2}$$

Typically, force-directed placers optimize a quadratic approximation of the HPWL:

$$\Phi_G(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{3.3}$$

From Equation 3.3, (x_i, y_i) represents the coordinates of cell i, and $w_{i,j}$ represents the net-weight of the connection between cells i and j.

In this chapter, a force-directed global placer in the spirit of SimPL [37], where $w_{i,j}$ is calculated by the Bound2Bound net model [76], is used along with a detailed placer derived from FastPlace-DP [58]. Briefly, SimPL is a flat, force-directed global placer. It maintains a lower-bound and an upper-bound placement and progressively narrows the displacement between the two to yield a final placement solution. The upper-bound placement is generated by applying *lookahead legalization (LAL)*, which is based on top-down geometric partitioning and non-linear scaling. The coordinates obtained from the upper bound placement are used as fixed points, which are connected to their corresponding cells via pseudo nets to provide spreading forces. The lower-bound placement is then generated by minimizing the quadratic objective in Equation 3.3.

3.3.1 Alignment groups

Definition 1. An alignment group $g_k \in G$ where $1 \leq k \leq |G|$, is an unordered subset of cells from V. An alignment direction $\vec{d_k}$ is a preferred placement direction of g_k where $0 \leq \vec{d_k} \leq 90$, with 0 representing horizontal and 90 representing vertical direction.

Essentially, an alignment group is a set of cells that need to be aligned in a certain direction (e.g., horizontally or vertically) during placement to optimize the datapath. In this chapter, it is assumed that the set of alignment groups G, and their alignment directions are given. Additionally, the collection of g_k with the same d_k value is pairwise disjoint.

Generally, g_k may correspond to bit-stacks in the datapath, but can be other elements such as cells connected to a single high-fanout net that improves through alignment, buffers that need careful placement to facilitate routing of large buses, or pipelining latches. For alignment directions, in this chapter, only horizontal and vertical directions are considered, which means $\vec{d_k} \in \{0, 90\}$. The above assumptions that alignment groups and directions can be given are valid and practical. One may use datapath extractors such as [88, 15, 65] to extract the alignment groups based on circuit properties. Alternatively, this information may come directly from logical descriptions of the netlist, or could be provided by designers. As an example, if designers are trying to structure the placement of latches, it is trivial for them to provide sets of latch names and their preferred placement directions.

3.3.2 Pseudo nets

Fixed-point generation followed by pseudo net insertion is a common method to apply spreading forces during iterative force-directed placement.

Definition 2. A pseudo net $c(f, i)$ is a weighted two-pin connection between a fixed-point f and a cell i in the circuit netlist. The pseudo net has a weight equal to $\epsilon \cdot w_{i,j}$, defined in [37], and does not exist in the circuit netlist.

Please note that, these nets are added for every movable cell in the design. In addition, existing pseudo nets are discarded at the end of the current iteration, and a new set is added to enforce spreading during the subsequent placement iteration. The pseudo net-weighting technique with

varying ϵ is described in [37, 38], and controls the rate of overlap removal during global placement. During early iterations, greater significance is given to interconnect minimization, while the relative cell ordering stabilizes. This is accomplished by starting with a small ϵ value and gradually increasing it through each iteration. This scheme provides flexibility to the placer during the early stages, while tightening the constraints to resolve overlap towards the end of global placement. The theoretical justification and extensions of the SimPL framework are provided in [38].

3.3.3 Alignment nets

To achieve better datapath alignment, one approach is direct manipulation of existing nets between the datapath cells. However, this approach interferes with other placement directives. Specifically, direct weighting manipulation of current nets disrupts timing-driven / power-driven placement and net-weighting for those cells. Hence, a new category of nets is introduced, referred as *alignment nets*, and defined as follows.

Definition 3. An alignment net s_k, where $1 \leq k \leq |G|$, is a weighted multi-pin connection among all cells in an alignment group g_k. For placement, this multi-pin alignment net is decomposed into 2-pin nets, where netweights are found by the Bound2Bound net model [76].

Alignment nets are created at the beginning of placement and remain persistent during the entire global and detailed placement stages. A skewed-netweight scheduling helps these nets align the cells within the corresponding

alignment group g_k inside the layout region. By applying alignment constraints to new nets s_k, prior directives relying on net-weighting continue to function as before.

3.4 Unified Placement Flow with Alignment Constraints

Figure 3.2 presents a unified structure-aware placement flow, referred to as SAPT, which simultaneously places both datapath and random logic. The shaded boxes represent the steps in a conventional force-directed placement flow and the white boxes represent enhancements to better handle datapath circuits. The key algorithmic components of SAPT from the flow diagram are as follows:

1. *Alignment Net Insertion:* Alignment nets are inserted to manipulate placements of a specific set of cells during global placement (Section 3.3.3).

2. *Target Skew Ratio Generation:* During placement, the alignment net-weights are modulated to optimize the datapath placement. A method to automatically generate the maximum netweights in the horizontal and vertical directions is developed to achieve a desired spread or span for an alignment net over the layout region. (Section 3.5.2).

3. *Skewed Weighting with Step Size Scheduling:* This describes the skewed net-weighting process applied to each alignment net s_k to gradually improve alignment along the datapath. The weight in a particular direction

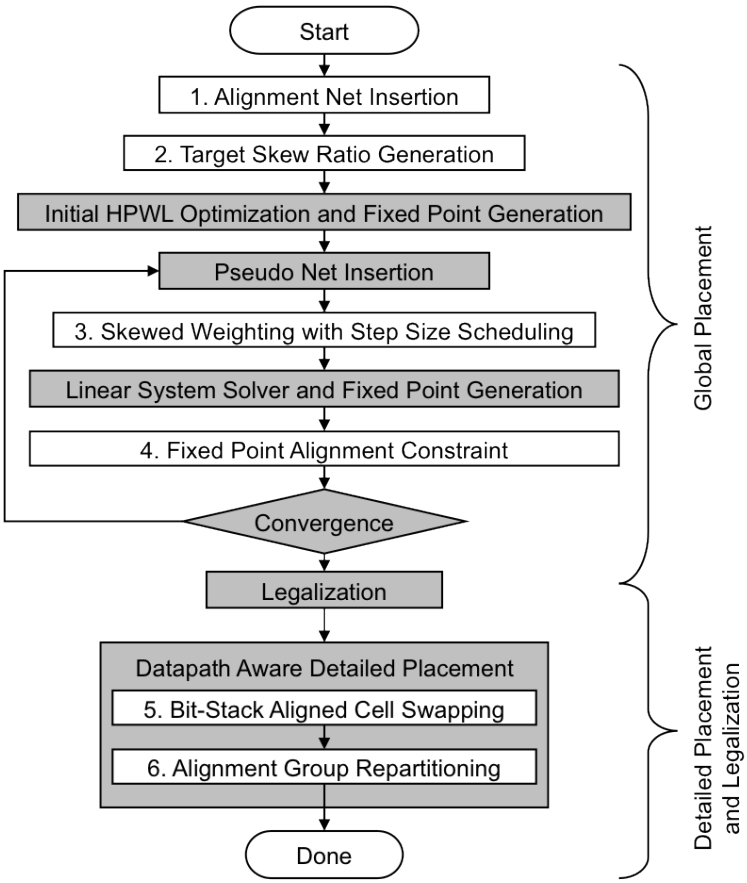


Figure 3.2: Proposed unified datapath-aware placement flow. The baseline components are shown in shaded boxes and the newly added datapath-aware components are shown in transparent boxes.

gradually increases during placement iterations until it reaches the target skew for that direction. (Section 3.5.1).

4. *Fixed-Point Alignment Constraint:* Force-directed global placement frameworks reduce cell overlap by using fixed-points and pseudo nets. When a placer is structure-unaware, this process leads to misalignment for dat-

apath logic. This section describes a process to modify the fixed-point locations to improve structured placements.

5. *Bit-Stack Aligned Cell Swapping:* Generating alignment during global placement does not guarantee that cells within the alignment group will remain aligned during detailed placement. The global cell swapping step of detailed placement is enhanced to maintain the alignment generated during global placement (Section 3.6.1).

6. *Alignment Group Repartitioning:* With datapaths, traditional HPWL metrics can at times fail to detect alignment improvements. This technique minimizes internal net cut values, potentially improving both HPWL and StWL metrics for all nodes in g_k along s_k (Section 3.6.2).

At each step, the modifications apply only to the defined alignment groups g_k (Section 3.3.1) leaving all other random-logic cells to be placed as they would before. Though in this work SimPL and FastPlace-DP are used as an example, the techniques can be adapted to other force-directed placement algorithms. Detailed description of the global and detailed placement techniques are presented in Sections 3.5 and 3.6 respectively.

3.5 Structure-Aware Global Placement

This section presents techniques for providing alignment constraints to cells during global placement. Skewed weighting with step size scheduling is

described followed by methods to force alignment by modifying fixed-point locations.

3.5.1 Skewed weighting with step size scheduling

In this section, a skewed weighting technique is introduced that encourages alignment of s_k in preferred placement directions $\vec{d}_k$. The high level idea is to apply gradually increasing net-weights for s_k only in orthogonal directions to $\vec{d}_k$. This application of higher net-weights in one particular direction (i.e., skewed weighting) increases corresponding costs (HPWL($\vec{x}$) or HPWL ($\vec{y}$)) of the quadratic objective. Consequently, the linear system solver generates compact placement in unpreferred directions to reduce the overall cost.

The rate of change of the weighting value increases slowly during the initial stages of global placement, increases rapidly during the middle stages, and slows again near the end of global placement. This is preferable to applying hard constraints (forced alignment) in the early stage of wirelength optimization, as it can disrupt the original optimization and often lead to a solution that suffers from sub-optimality in terms of overall wirelength.

3.5.1.1 Step size scheduling

Let n be the current global placement iteration and M be its upper bound,[4] and $p(n)$ is defined as the alignment weight schedule function for each

[4] M is typically set to 50 [37].

iteration n. The function $p(n)$ is adjusted as follows:

$$p(n) = \begin{cases} \frac{8n^2}{M^2} & 0 \leq n < \frac{M}{4} \\ 1 - \frac{8(n-\frac{M}{2})^2}{M^2} & \frac{M}{4} \leq n \leq \frac{3M}{4} \\ \frac{8(n-M)^2}{M^2} & \frac{3M}{4} < n \leq M \end{cases} \tag{3.4}$$

To avoid imposing hard constraints during the initial stages of global placement, $p(n)$ gradually increases during the initial iterations and to minimize large constraint changes during the final stages, the function decreases towards zero at the last iteration. This function is also used in [33] to model cell density, but it serves a *completely different purpose* here as a scheduling function.

Using $p(n)$, Equation (3.5) displays the skewed monotonically increasing weighting parameters γ^n and δ^n for alignment net s_k. Using $p(n)$ directly generates very large weighting steps therefore a constant scaling factor β is added. This parameter is left default throughout all placement runs. Let $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ be the directional unit vectors and $\sigma^2_{x,y}$ the nth iteration's variance in either the x or y direction. Finally, the modified placement equation is shown in Equation (3.6). For non-alignment nets, $\delta_{i,j} = 0$ and $\gamma_{i,j} = 0$.

$$\begin{aligned} \gamma^n &= \gamma^{n-1} + \hat{\mathbf{y}} \cdot \overrightarrow{\mathbf{d_k}} * \beta * p(n) * \sigma^2_x(n) \quad \text{where } \gamma^0 = 1 \\ \delta^n &= \delta^{n-1} + \hat{\mathbf{x}} \cdot \overrightarrow{\mathbf{d_k}} * \beta * p(n) * \sigma^2_y(n) \quad \text{where } \delta^0 = 1 \end{aligned} \tag{3.5}$$

$$\begin{aligned} \Phi_G(\vec{x}, \vec{y})^n = \sum_{i,j} [&(\gamma^n_{i,j} + w_{i,j})(x_i - x_j)^2 + \\ &(\delta^n_{i,j} + w_{i,j})(y_i - y_j)^2] \end{aligned} \tag{3.6}$$

Figure 3.3 displays the average change in wirelength between fixed scheduling and the bell-shaped step-size scheduling using different values of β. It shows that for $0.2 < \beta < 0.9$, the bell-shaped step-size scheduling yields on average, better HPWL as well as StWL. In this implementation, $\beta = 0.5$ is used based on empirical data.

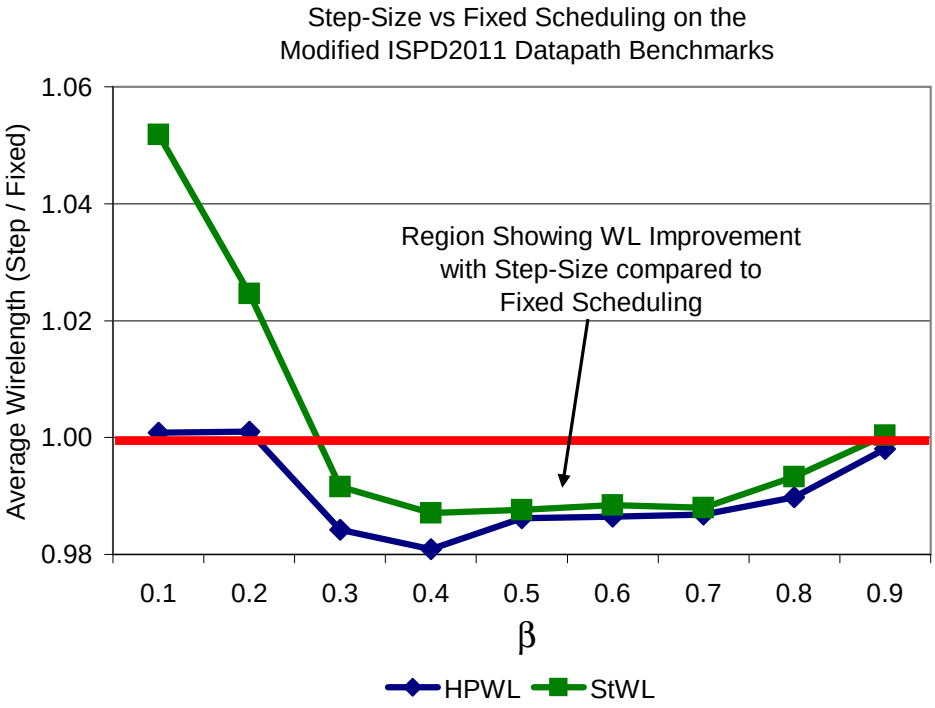


Figure 3.3: Average wirelength improvement using the bell-shaped step-size scheduling function on the ISPD2011 Datapath Benchmark Suite at different values of β.

3.5.2 Target skew ratio generation

Section 3.5.1 presents a method to systematically increase the skewed weight for an alignment net by either increasing $\gamma_{i,j}$ or $\delta_{i,j}$ as placement evolves.

In the preliminary version of this chapter [89], the maximum skew for γ^n or δ^n were provided as inputs, primarily based on the designers‘ discretion. This section avoids this shortcoming by presenting key insights and a method for automatic selection of the maximum skew values. *Target skew ratio* is defined as follows.

Definition 4. Target skew ratio is the maximum ratio γ^n/δ^n of the net-weighting in the x-direction and y-direction for an alignment net during global placement.

Qualitatively, the multiplier value defines how quickly the skewed weight grows with each consecutive global placement iteration. The target skew ratio defines the maximum ratio for a particular net and too large of a target skew negatively impacts the overall wirelength. As an example, given a vertically aligned alignment net, during the initial global placement iteration $\gamma^0 = 1$ and $\delta^0 = 1$. However, by the end of global placement iteration n, the X-direction weight would have increased to $\delta^n = \delta^0 * sk_i$ causing the cells within the alignment group to become aligned vertically. Key insights are now outlined for effective target skew multiplier selection and illustrated using Figure 3.4. In this figure, the red alignment net is shown connecting each of the datapath cells within an alignment group and the dashed lines indicate net connections among each cell within the alignment group and other cells outside the alignment group.

The first key insight is that the alignment net-weight must overcome the interconnect forces imposed by the nets connected among cells within the alignment group. In other words, the force pulling the alignment group into

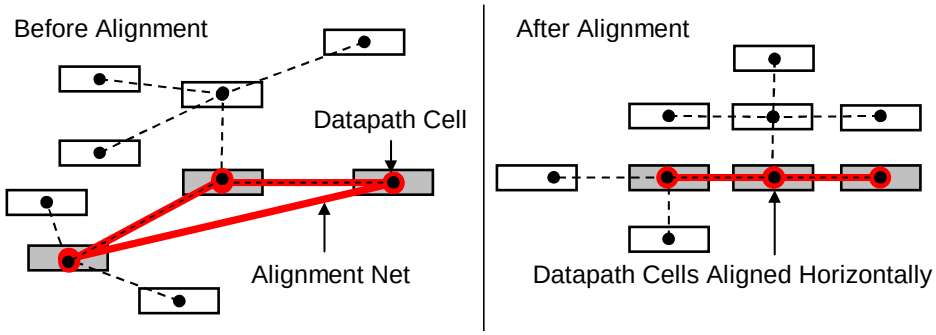


Figure 3.4: Horizontal skewed weight force example.

alignment must be greater than the force pulling the cells out of alignment. Equation (3.7) relates the force from internal pin connections on the alignment net versus the force from external pin connections on the datapath cells.

$$T = g^p \tag{3.7}$$

This chapter solves for the p_i exponent value for a particular alignment net i such that the internal force is always greater than the external force. The variable T equals the total external pin cardinality for an alignment group, which is the sum of all the pins for each cell. This includes pins with a net connecting cells within an alignment group and pins connecting to cells outside of the alignment group. The variable g equals the total number of internal connections which are pins connected to the alignment net. The resulting p value relates the internal connections on the alignment net to the external connections among the cells and a larger p value implies more external connections necessitating a higher skewed weight multiplier. Conversely, a smaller p value indicates fewer external connections thus requiring a smaller multiplier.

The second key insight for automatic target skew ratio selection is consideration of additional interconnect forces from alignment nets themselves.

In the degenerate case, the unpreferred direction retains a weight of zero and this consideration can be ignored. However, for the general case, the unpreferred direction is not zero. The primary reason for a nonzero unpreferred weighting would be to address compaction of the bit-stack. Though this chapter does not explicitly consider this case, there could clearly be cases where compaction can improve overall placement quality. Thus, the alignment nets are employed to encourage very compact placement in unpreferred directions, however, they pull the connected cells together even in $\vec{d_k}$ when the unpreferred weight is not zero. The HPWL model naturally clumps the cells in alignment nets together in this general case, and therefore the target skew ratio must be properly adjusted to offset this side effect. A key observation is that the required skew ratio should be linearly increased as the cardinality of an alignment group g_i increases. Thus, the second component of the target skew multiplier is defined as follows: $\alpha * |g_i|, \alpha > 1$ where α is a placer specific constant that insures for a two-pin net, the two cells are always correctly aligned. For example, if aligning two cells horizontally, the Y-weight must be at least α times larger than the X-weight to guarantee the cells align horizontally.[5] For this chapter, $\alpha = 1.25$. This constant is then multiplied by the total number of cells in the alignment group. Finally, taking both components into account, the skewed weight multiplier is calculated as shown in Equation

[5]This parameter is set when placing only two cells.

(3.8).

$$\xi_i = (1 + log_g(T)) * \alpha * |g_i| \tag{3.8}$$

3.5.3 Fixed-point alignment

Force-directed global placement frameworks use fixed-points and pseudo nets to discourage cell overlap. By gradually perturbing the unconstrained linear system solver, global placement iterations progressively generate placements with less overlap. In SimPL [37], at each global placement iteration, lookahead legalization (LAL) generates fixed points that are connected to their corresponding movable cells with two-pin pseudo nets. In the subsequent global placement iteration, these pseudo nets exert pulling forces and reduce the amount of cell overlap. For datapath logic, the LAL step and subsequent pseudo net insertion step cause misalignment within the bit-stack requiring a constraint forcing alignment which minimizes wrong-way perturbations in the bit-stack.

Imposing fixed-point alignment constraint during global placement is achieved in two steps. First, LAL generates fixed-point locations for all movable cells. Second, fixed-point locations are modified for all cells in alignment group g_k. The modified fixed-point location for cell i is denoted as $\eta^n_{k,i}$ for the nth iteration, which is computed as follows:

$$\eta^n_{k,i} = \begin{cases} (x^n_i, \frac{\Sigma_{j=1,\ldots,|g_k|} y^n_j}{|g_k|}), & \text{if } d_k = 0 \\ (\frac{\Sigma_{j=1,\ldots,|g_k|} x^n_j}{|g_k|}, y^n_i), & \text{if } d_k = 90 \end{cases}$$

An example of the modified fixed-point locations and corresponding pseudo

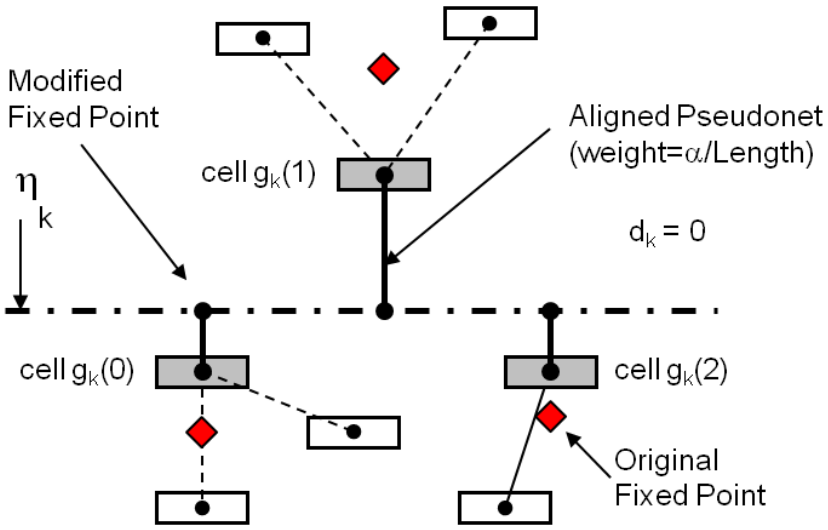


Figure 3.5: Example of a fixed-point alignment constraint for a horizontal bit-stack. Lookahead legalization generates new zero area fixed-points and the locations of these points are modified to be in alignment with η_k^n.

nets for a horizontal datapath is shown in Figure 3.5. In this example, there are three gray cells, $g_k(0:2)$ in one alignment group g_k. The other cell connections are shown with the dashed lines connected to the hollow cells. For random logic cells, the fixed-point locations will be determined solely based on the LAL step. For alignment groups, those locations are modified based on the arithmetic mean parallel to the preferred placement direction $\vec{d_k}$.

3.6 Structure-Aware Detailed Placement

Prior detailed placement techniques can disrupt the structure generated during their wirelength-optimization processes as they are structure un-

aware. This section presents two detailed placement techniques to maintain the aligned placement obtained by global placement. The impact of these techniques is reported in Section 3.7.

3.6.1 Bit-stack aligned cell swapping

This technique extends global cell swapping from [58] for nodes within each g_k by modifying the "swap region" while keeping the overlap penalty the same. Assuming all cells in the layout region are fixed except for cell j, the "swap region" based on the median idea from [23], is the location where the wirelength for cell j is improved if it is swapped with a cell k located in the swap region. This technique seeks cells to swap between the current location of cell j and all cells within the swap region. If swapping improves HPWL, the cell locations are updated.

This chapter, unlike [58], bounds the swap region perpendicular to $\vec{d_k}$. Specifically, for each net $p \in E$, the left, right, lower and upper edges of the bounding box are: ($x_l[p]$,$x_r[p]$,$y_l[p]$,$y_u[p]$) and the x^{opt} and y^{opt} from [23] is the median of the x series ($x_l[1]$, $x_r[1]$, $x_l[2]$, $x_r[2]$, ...) and y series ($y_l[1]$, $y_u[1]$, $y_l[2]$, $y_u[2]$, ...), respectively. Because the number of elements is generally even, the x^{opt} and y^{opt} becomes a region with bounding box (x_l^{opt},y_l^{opt} , x_r^{opt},y_u^{opt}). Equation (3.9) finds the modified swap region for a cell within an alignment group with $\vec{d_k} = 0$ while Equation 3.10 finds it for a cell within an alignment

group with $\vec{d_k} = 90$.

$$\begin{gathered} x_l^{opt}, min_y(g_k) - var_y(g_k) \\ x_r^{opt}, max_y(g_k) + var_y(g_k) \\ when \qquad (\overrightarrow{\mathbf{d_k}} = 0) \end{gathered} \tag{3.9}$$

$$\begin{gathered} min_x(g_k) - var_x(g_k), y_l^{opt}, \\ max_x(g_k) + var_x(g_k), y_u^{opt} \\ when \qquad (\overrightarrow{\mathbf{d_k}} = 90) \end{gathered} \tag{3.10}$$

Figure 3.6 contrasts the original potential swap region and the structure-aware swap region. In the original example from Figure 3.6(a), the swap region for cell j, based on the HPWL would cause j to move out of line with $\vec{d_k}$ for that group, thus disrupting the alignment. To the contrary, Figure 3.6(b) demonstrates that the swap region is shifted down to maintain alignment for that group.

3.6.2 Alignment group repartitioning

The second detailed placement technique is a top-down recursive repartitioning for each g_k along alignment net s_k, referred to as alignment partitioning shown in Algorithm 1. With datapaths, traditional HPWL metrics can at times fail to detect alignment improvements. This technique minimizes internal net cut values potentially improving overflow without impacting HPWL and StWL metrics for all nodes in g_k along s_k. The base cut algorithm is from [36], but there are a couple of key differences to the repartitioning method. First, the swap is only accepted when the HPWL after the swap is less than

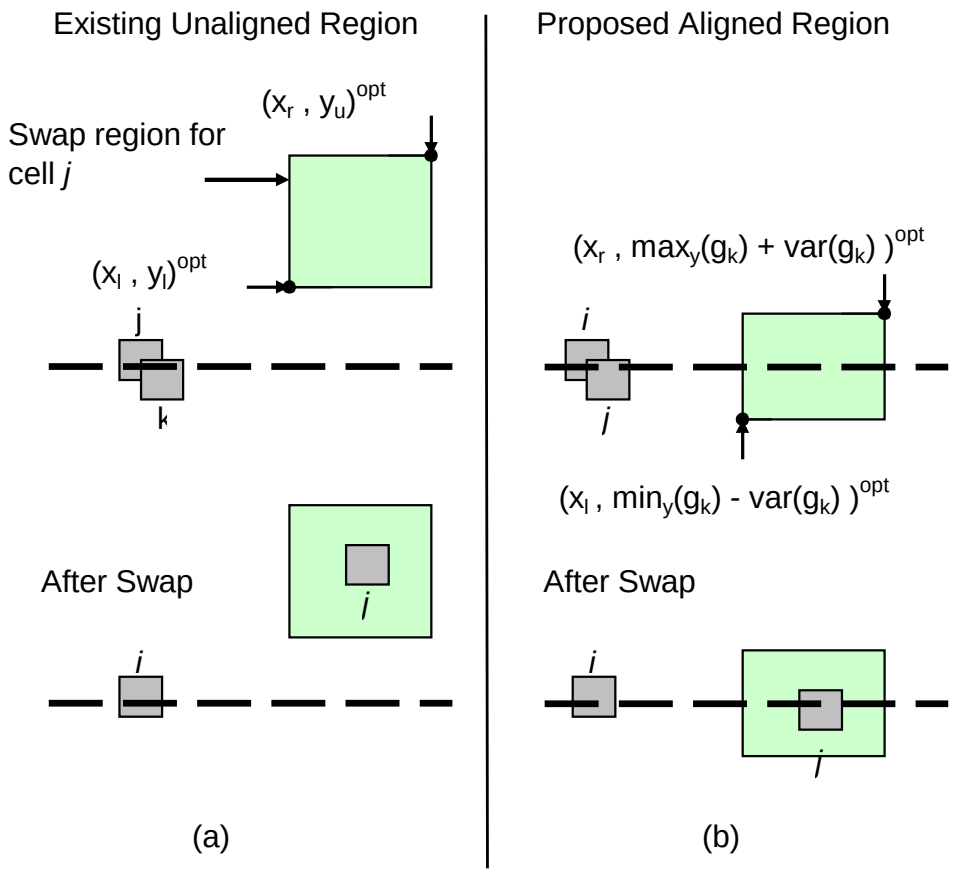


Figure 3.6: Swap region shift for cell j when the alignment direction is parallel to the x-axis. The upper y coordinate location is defined by cell i plus the variance between cell i and cell j.

or equal to the HPWL before the swap. Second, the initial median is the midpoint between the nodes. All nodes in g_k with values less than the median go in one partition, while the other nodes go in the other partition.

For each defined alignment group in the design, the algorithm calculates the median or middle point of s_k. Once a median point has been identified, the algorithm partitions all nodes connected to the alignment group using KL

Inputs : A netlist N, alignment groups g_k, alignment nets s_k
Output : A reordered set of nodes g_k along each s_k
function bipartition(g_k):
1: determine median cell within g_k
2: partition g_k into sets A and B with median from 1
3: evaluate initial HPWL of g_k
4: call KL Partitioning with partitions A,B
5: evaluate HPWL $\forall$ nodes $\in A, B$ on updated partition
6: if (HPWL of update partition is $<$ the initial HPWL) g_k = new partition
7: else revert to initial A and B partitions
8: if($|A| >$ minimum partition size) bipartition(A)
9: if($|B| >$ minimum partition size) bipartition(B)
10: return g_k

Algorithm 1: Alignment Group Repartitioning

partitioning [36]. The partitioning solution is made HPWL-aware by evaluating the KL solution for HPWL changes. If the HPWL increases, the solution is discarded and KL evaluates a new solution for a higher net cut value that does not cause an increase in HPWL or total net cut. Once a partition has been selected, the design is legalized and the loop at that partition level is complete. The algorithm continues to hierarchically break, using recursive repartitioning, each alignment group into smaller partitions until a predefined minimum partition size is met. By minimizing cut value, improved alignment and routability is possible. As an example, consider Figure 3.7(a) with median location m_i, alignment group s_i in $Row(j)$. In this placement, the initial cut value across m_i is three. After swapping nodes b_i and a_i, shown in Figure 3.7(b), the total net cut value along m_i is reduced to one.

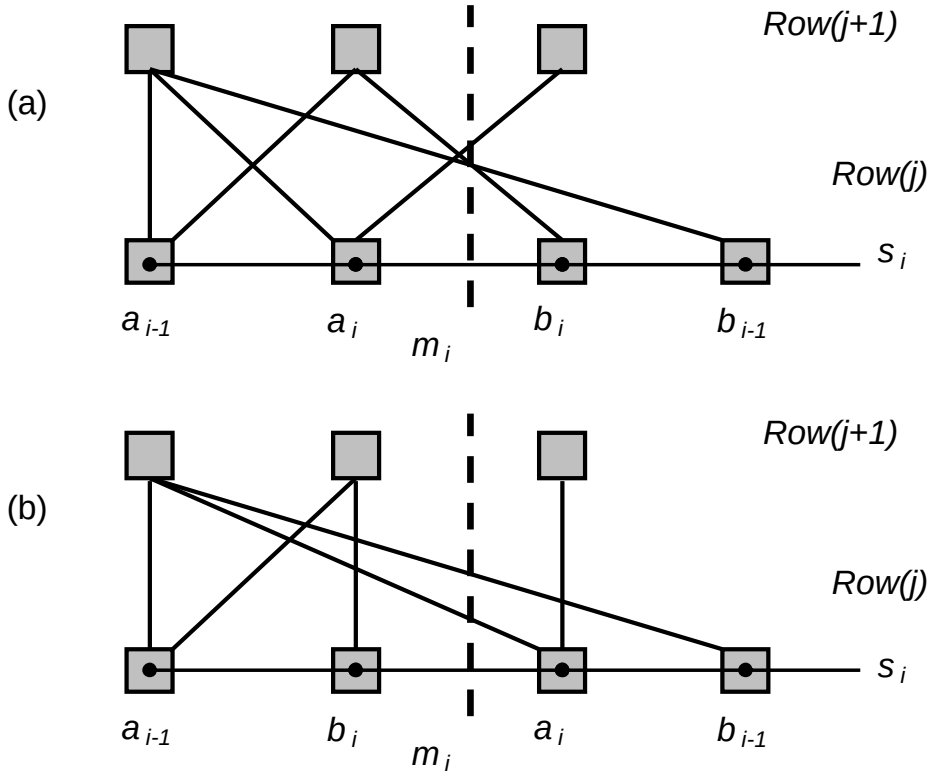


Figure 3.7: Group repartitioning example swapping the positions of cell a_i and b_i for an improved net cut.

3.7 Experimental Results

The SAPT implementation is in C++, built on the SimPL [37] global placement and FastPlace-DP [58] detailed placement frameworks. It is compatible with the Bookshelf format and requires an additional datapath definition file as input. This file, loaded prior to global placement, includes each cell of the alignment group and the group's direction. Since this chapter focuses on the placement solution, each datapath was manually defined for improved experimental control. SAPT is empirically validated with two design styles, (*i*) the Modified ISPD (MISPD) 2011 Datapath Benchmark Suite [91] and (*ii*) industrial *hybrid* designs. All experiments were run on a 3.2 GHZ Intel(R) Xeon(R) X5672 Linux workstation with 96 GB of memory. Table 3.2 displays the parameter settings used for all experiments.

Table 3.2: Parameter values set used for experiments.

Parameter	Value	Section	Description
g_k	User Defined	3.3.1	Alignment Group
d_k	User Defined	3.3.3	Alignment Group Direction
β	0.5	3.5.1.1	Constant Scaling Factor
α	1.25	3.5.2	Constant force required to align two cells

To independently quantify the effectiveness of the global and detailed placement techniques, results are provided for the following three flows:

- SAPTsg + FP-DP: Only the skewed weighting with step size scheduling technique (Section 3.5.1) followed by the default FastPlace-DP detailed placer.

- SAPTgp + FP-DP: The structure-aware global placement followed by the default FastPlace-DP detailed placer.

- SAPTgp + SAPTdp: The structure-aware global and detailed placement techniques.

The SAPT placer is compared against state-of-the-art academic placers:[6] CAPO v10.2 [68], mPL6 [6], NTUPlace3 v7.10.19 [12], Rooster [67], FastPlace v3.0 [80], Dragon v3.01 [87] and SimPL [37]. All placers were supplied a target density of 1.0 to achieve the best wirelength, of which the placers are capable. This configuration corresponds to manual placement, where movable cells are often packed to optimize wirelength with no local whitespace injected. All HPWL and StWL results are reported on legalized placements using the coalesCgrip [74] tool.

3.7.1 Benchmark circuits

Tables 3.3 and 3.4 provide the circuit characteristics. Of note is the number of alignment groups, g_k, and the datapath ratio in each design, where the datapath ratio is defined as the ratio of alignment group cells to random-logic cells. Though the hybrid designs are on the smaller side, they are state-of-the-art industrial circuits, and pose challenges for designers as they contain regular structures intermixed with other random logic. This makes it difficult

[6]The recent structure-aware placer from [14] does not provide StWL results for a direct comparison. As of Aug 2012, a request has been made to the authors for the binary.

to place the datapath logic separately from the random logic. The ISPD 2011 Datapath Benchmark Suite contains two datapath circuits, each with eight different utilizations to examine the ability of automatic placers to generate placement solutions at different densities on highly regular structures. In this chapter, *the benchmarks were modified to make all the latches movable compared to the fixed latch placement in the original work.* All logical connections and I/O pin locations remain the same. The SAPT placer is compared against other placers on the Modified ISPD 2011 (MISPD) Datapath Benchmarks, as *unfixed latch* placement is more challenging and often indicative of hybrid industrial designs.

Table 3.3: Benchmark circuit statistics. Datapath ratio is calculated as the total number of datapath cells divided by the total number of cells.

	ISPD2011 Datapath Benchmarks	
	Benchmark A	Benchmark B
Total node count	160416	152668
Total pin count	637984	653116
Total net count	157849	148682
Alignment groups g_k	1425	1932
Datapath ratio	0.92	0.85

Table 3.4: Hyrbid circuit statistics. Datapath ratio is calculated as the total number of datapath cells divided by the total number of cells.

	Industrial Hybrid Designs					
Hybrid	C	D	E	F	G	H
Total node count	17922	55387	83802	263906	194271	62133
Total pin count	64078	94682	130000	397652	343727	21124
Total net count	16874	14458	16422	53884	62145	101582
Alignment groups g_k	35	110	60	131	82	28
Datapath ratio	0.01	0.012	0.008	0.007	0.018	0.021

3.7.2 Wirelength results on the MISPD 2011 Datapath Benchmark Suite

Tables 3.5, 3.6, 3.7 and 3.8 compare the total HPWL and total StWL on the MISPD 2011 Datapath Benchmark Suite. To clarify, the total wirelength calculation includes both random-logic and datapath-logic nets in the design. All results are the ratio of the total wirelength of the automatically placed solution to the total wirelength of the manually designed placement as described in [92]. Tables 3.5 and 3.6 indicate that the total HPWL is within a few percentage points of the best HPWL solution for each of the benchmarks. In fact, the best HPWL on 7/8 of the variants for benchmark A are obtained with the SAPT placer.

Tables 3.7 and 3.8 demonstrate that the SAPT placer (SAPTgp + SAPTdp) obtains significantly better StWL than other automatic placers for all benchmarks. For the benchmark A variants, the StWL of all the other placers is greater than 1.5× the manually designed solution. For the benchmark B variants, the results are greater than 2.0× the manually designed solution. Alternatively, for benchmark A, the skewed weighting with step step size scheduling technique (SAPTsg + FP-DP) was able to obtain a solution that is 1.35× the manually designed solution at (at 79% utilization). Both global placement techniques (SAPTgp + FP-DP) improved the solution to 1.31× the manually designed solution (also at 79% utilization). Additionally, the detailed placement techniques (SAPTgp + SAPTdp) were able to further reduce the gap to 1.26×. The SAPT placer on benchmark B also significantly

outperform prior placers, with SAPTsg + FP-DP achieving 1.49×, SAPTgp + FP-DP achieving 1.48× and SAPTgp + SAPTdp achieving 1.46× the StWL of the manually designed solution.

Table 3.5: Total HPWL ratio comparison on the MISPD 2011 Datapath Benchmark A variants on legalized placements. The ratios are computed with respect to the manually placed solution.

	ISPD 2011 Datapath Benchmark A: Total HPWL							
Utilization	94	91	89	86	84	82	79	77
CAPO	**1.05**	1.04	1.04	1.04	1.03	1.02	1.06	1.03
mPL6	1.17	1.19	1.22	1.14	1.16	1.2	1.17	1.16
NTUPlace3	1.22	1.19	1.16	1.19	1.15	1.19	1.23	1.26
Dragon	1.49	1.58	1.63	1.6	1.51	1.62	1.66	1.6
FastPlace3	1.42	1.5	1.53	1.54	1.53	1.67	1.7	1.75
Rooster	**1.05**	1.04	1.04	1.04	1.03	1.02	1.06	1.03
SimPL	1.08	1.07	1.06	1.07	1.05	1.06	1.05	1.04
SAPTsg + FP-DP	1.09	1.11	1.07	1.05	1.06	1.05	1.04	1.04
SAPTgp + FP-DP	1.07	1.04	1.05	1.03	1.03	1.02	1.02	1.01
SAPTgp + SAPTdp	1.06	**0.99**	**1.03**	**1.02**	**1.02**	**1.01**	**1.01**	**0.98**

Figure 3.8(a) displays the placement solution from SimPL on benchmark A. In this figure, a random selection of cells in alignment groups are plotted with a brown or blue line connecting them. Clearly, the cell placement is not aligned. In the manually-placed solution, these cells are aligned, either vertically or horizontally depending on the alignment group, yielding shorter wirelength. Figure 3.8(b) shows the placement solution generated using the SAPT placer, with the same set of alignment groups selected as Figure 3.8(a), highlighting significant improvement in the alignment of g_k.

Table 3.6: Total HPWL ratio comparison on the MISPD 2011 Datapath Benchmark B variants on legalized placements. The ratios are computed with respect to the manually placed solution.

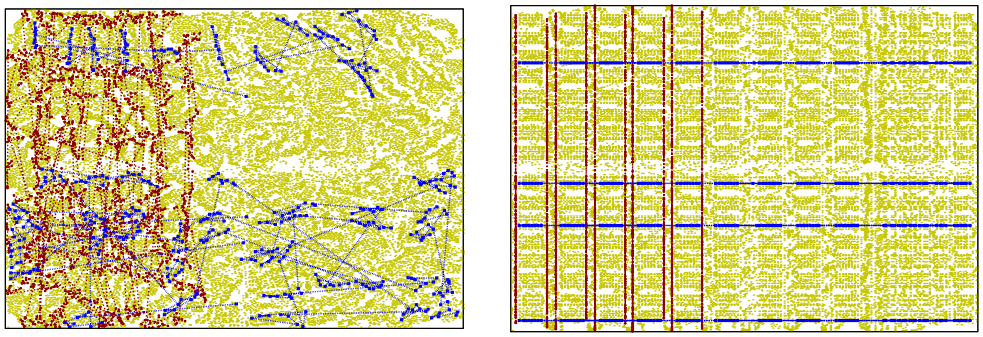
	ISPD 2011 Datapath Benchmark B: Total HPWL							
Utilization	95	93	91	89	86	84	81	79
CAPO	1.2	1.18	1.17	**1.12**	**1.13**	**1.13**	1.14	**1.12**
mPL6	1.64	1.86	1.72	1.64	1.65	1.65	1.78	1.78
NTUPlace3	1.25	1.19	1.17	1.15	1.16	1.15	**1.12**	1.15
Dragon	1.4	1.4	1.35	1.32	1.32	1.3	1.31	1.31
FastPlace3	1.69	1.66	1.73	1.71	1.77	1.86	1.77	1.87
Rooster	**1.15**	**1.15**	**1.13**	**1.12**	1.15	**1.13**	1.13	1.13
SimPL	1.23	1.22	1.21	1.2	1.17	1.16	1.16	1.15
SAPTsg + FP-DP	1.22	1.2	1.18	1.17	1.17	1.16	1.16	1.15
SAPTgp + FP-DP	1.21	1.2	1.17	1.16	1.16	1.16	1.16	1.15
SAPTgp + SAPTdp	1.21	1.19	1.17	1.16	1.15	1.15	1.14	1.15

Table 3.7: Total StWL ratio comparison on the MISPD 2011 Datapath Benchmark A variants on legalized placements. The ratios are computed with respect to the manually placed solution. Numbers in bold are the best automated placement results published for these benchmarks.

	ISPD 2011 Datapath Benchmark A: Total StWL							
Utilization	94	91	89	86	84	82	79	77
CAPO	1.94	1.94	1.91	1.93	1.9	1.9	1.8	1.9
mPL6	2.16	2.14	2.16	2.08	2.1	2.12	2.11	2.09
NTUPlace3	2.23	2.18	2.15	2.15	2.11	2.16	2.19	2.09
Dragon	2.37	2.44	2.53	2.48	2.36	2.48	2.56	2.43
FastPlace3	2.45	2.53	2.56	2.59	2.56	2.71	2.75	2.79
Rooster	1.94	1.93	1.91	1.91	1.92	1.91	1.82	1.95
SimPL	1.82	1.83	1.8	1.81	1.78	1.78	1.78	1.75
SAPTsg + FP-DP	1.43	1.45	1.38	1.37	1.37	1.36	1.35	1.35
SAPTgp + FP-DP	1.4	1.35	1.36	1.33	1.33	1.32	1.31	1.31
SAPTgp + SAPTdp	**1.35**	**1.28**	**1.31**	**1.31**	**1.28**	**1.28**	**1.26**	**1.28**

Table 3.8: Total StWL ratio comparison on the MISPD 2011 Datapath Benchmark B variants on legalized placements. The ratios are computed with respect to the manually placed solution. Numbers in bold are the best automated placement results published for these benchmarks.

	ISPD 2011 Datapath Benchmark B: Total StWL							
Utilization	95	93	91	89	86	84	81	79
CAPO	2.4	2.4	2.38	2.35	2.36	2.36	2.35	2.32
mPL6	2.94	3.29	3.06	3.01	2.97	2.95	3.2	3.21
NTUPlace3	2.66	2.48	2.47	2.44	2.44	2.44	2.32	2.44
Dragon	2.91	2.87	2.84	2.8	2.79	2.77	2.75	2.74
FastPlace3	3.73	3.58	3.78	3.79	3.97	4.13	3.96	4.14
Rooster	2.4	2.39	2.35	2.33	2.38	2.34	2.34	2.32
SimPL	2.27	2.3	2.25	2.24	2.23	2.19	2.24	2.22
SAPTsg + FP-DP	1.61	1.57	1.55	1.52	1.52	1.51	1.5	1.49
SAPTgp + FP-DP	1.59	1.56	1.54	1.5	1.51	1.5	1.5	1.48
SAPTgp + SAPTdp	**1.58**	**1.55**	**1.52**	**1.49**	**1.49**	**1.48**	**1.48**	**1.46**



(a) (b)

Figure 3.8: Forty structured bit-stacks are randomly chosen to show the alignment impact of SAPT on benchmark A. (a) is generated by SimPL[37] whereas (b) is generated by SAPT. Movable cells are shown lightly shaded while the cells in the alignment group are shown dark. Note that cells that are not defined by alignment groups become aligned as well and form a regular structure (b).

3.7.3 Wirelength results on the hybrid designs

Tables 3.9 and 3.10 give the normalized total HPWL and StWL results on the hybrid designs. All results are normalized to the SAPT results (SAPTgp + SAPTdp). As before, SAPT significantly improves StWL on the hybrid designs. Though the HPWL results are similar, SAPT obtains an improvement in StWL between 1% and 13%. For these designs, the StWL improvement is significant, considering the fact that the percentage of datapath logics within the designs is less than 3%.

By providing alignment constraints to portions of the datapath, it is observed that other neighboring cells also become aligned during the iterative placement process. The alignment constraints provide hints, directing the placer in the correct gradient. These hints help to overcome local optima, driving placement towards a more globally optimal solution.

3.7.4 Routing congestion results

To empirically validate the claim that StWL accurately approximates routability, two congestion metrics are reported on all benchmark circuits. It is noteworthy that the SAPT placer is congestion unaware, and that the advantage in routing congestion is simply a by-product of improved placement solutions.

Tables 3.11 and 3.12 display the total overflow (TOF)$\times(1e+5)$, as defined in the ISPD 2011 routability-driven placement contest [81] for the datapath benchmark circuits. All reported overflow values are from the official

Table 3.9: Normalized total HPWL and StWL comparison on the hybrid designs C, D and E. The wirelengths are normalized to the SAPT results.

	Hybrid C		Hybrid D		Hybrid E	
	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL
CAPO	1.13	1.26	1.17	1.32	1.12	1.27
mPL6	1.05	1.15	1.02	1.14	1.2	1.32
NTUPlace3	0.95	1.1	0.95	1.13	0.99	1.19
Dragon	1.1	1.2	2.11	2.04	1.32	1.38
Rooster	1.1	1.19	1.16	1.32	1.26	1.23
FastPlace3	0.95	1.04	0.96	1.16	1.22	1.3
SimPL	1.02	1.1	0.97	1.16	1.03	1.12
SAPTsg + FP-DP	1.05	1.08	1.06	1.07	1.07	1.05
SAPTgp + FP-DP	1.02	1.06	1.03	1.03	1.04	1.02
SAPTgp + SAPTdp	1	**1.00**	1	**1.00**	1	**1.00**

Table 3.10: Normalized total HPWL and StWL comparison on the hybrid designs F, G and H. The wirelengths are normalized to the SAPT results. The Dragon placer was unable to complete for Hybrid G.

	Hybrid F		Hybrid G		Hybrid H	
	Total HPWL	Total StWL	Total HPWL	Total StWL	Total HPWL	Total StWL
CAPO	1.19	1.17	1.25	1.23	1.23	1.21
mPL6	1.37	1.3	1.21	1.19	1.09	1.10
NTUPlace3	1.3	1.3	1.02	1.05	1	1.01
Dragon	1.29	1.24	-	-	1.22	1.20
Rooster	1.03	1.25	1.03	1.05	1.26	1.23
FastPlace3	1.17	1.14	1.11	1.1	1.04	1.03
SimPL	1.04	1.04	1.01	1.01	1.02	1.02
SAPTsg + FP-DP	1.04	1.05	1.05	1.08	1.08	1.06
SAPTgp + FP-DP	1.03	1.02	1.02	1.04	1.02	1.03
SAPTgp + SAPTdp	1	**1.00**	1	**1.00**	1	**1.00**

contest evaluation script. As Tables 3.11 and 3.12, SAPT produces the smallest overflow across all testcases. For benchmark A, SAPT produced a routable

placement solution with *zero* (0) overflow for all but two of the variations. For benchmark B, SAPT improves total overflow by 6.7×, 23.54×, 14.44×, 11.56×, 14.3×, and 10.36× versus SimPL, FastPlace3.0, Dragon, NTUPlace3, mPL6, and CAPO respectively. The second congestion metric is the peak weighted congestion (PWC) as used in the DAC 2012 routability-driven placement contest [82]. This metric calculates the weighted average ACE(x) congestion [95], of the top x% congested g-edges. The smaller the PWC value, the more routable the design. As with the overflow metric, in designs with significant structure, SAPT produces significantly more routable placements than the compared placers. Tables 3.13 and 3.14 display the PWC for each of the datapath benchmarks. As the resluts show, the proposed approach significantly reduces the peak weighted congestion for all variations.

Tables 3.15 and 3.16 displays the TOF using the ISPD 2011 contest router (coalesCgrip [74]) and the PWC using the DAC 2012 contest router (BFG-R [29]) for the hybrid designs. SAPT generated the best PWC results for Hybrids D, E, and G, and was comparable to the best results on the remaining designs. Additionally, SAPT generated the best TOF solution for three Hybrid designs D, E, and H. Though SAPT is not a congestion-aware placer, the significant improvement in routing congestion indicates the strong correlation between StWL quality and congestion.

Table 3.11: The Total Overflow (x 1e+5) using the router and evaluation script from the ISPD 2011 routability-driven placement contest on the MISPD 2011 Datapath Benchmark A variants on legalized placements. Routing resources are extracted from current 22nm technology node.

	ISPD 2011 Benchmark A: Routing Overflow							
Utilization	94	91	89	86	84	82	79	77
CAPO	2.29	2.17	1.72	1.83	1.84	1.68	1.1	2.18
mPL6	4.66	4.38	4.44	3.4	3.38	3.65	6.03	5.02
NTUPlace3	5.54	5.12	4.63	5.19	4.92	5.63	6.03	5.02
Rooster	0.79	1.08	1.22	1.02	1.07	1.64	1.31	1.16
FastPlace3	7.23	8.1	8.72	9.08	8.8	10.4	11.8	12.1
SimPL	1.28	1.28	1.22	0.98	0.87	0.87	0.85	0.77
SAPTgp + SAPTdp	0.0014	0.038	0	0	0	0	0	0

Table 3.12: The Total Overflow (x 1e+5) using the router and evaluation script from the ISPD 2011 routability-driven placement contest on the MISPD 2011 Datapath Benchmark B variants on legalized placements. Routing resources are extracted from current 22nm technology node.

	ISPD 2011 Benchmark B: Routing Overflow							
Utilization	95	93	91	89	86	84	81	79
CAPO	9.16	7.28	7.05	6.68	7.17	7.01	7.13	6.98
mPL6	12.7	16.4	14	13.6	12.8	12.6	15.3	15.3
NTUPlace3	10.2	8.41	8.3	8.09	8.92	9.07	8.21	9.92
Rooster	8.11	7.28	6.88	6.72	-	-	-	-
FastPlace3	20.8	19.3	21.6	21.7	23.7	25.5	23.5	25.6
SimPL	5.98	6.24	5.65	5.49	5.26	4.85	5.21	5.25
SAPTgp + SAPTdp	0.88	0.7	0.55	0.43	0.67	0.58	0.6	0.58

3.7.5 Runtime results

Runtime results on the Hybrid designs are shown in Table 3.17. Both SimPL and FastPlace3.0 were similar with FastPlace3.0 faster on larger designs. The SAPT placer performed very competitively compared to the other

Table 3.13: The peak weighted congestion (PWC) using the BFG-R [29] router and evaluation script from the DAC 2012 routability-driven placement contest on the MISPD datapath benchmark A variants on legalized placements. Routing resources are extracted from current 22nm technology node.

	ISPD 2011 Datapath Benchmark A: Average PWC							
Utilization	94	91	89	86	84	82	79	77
CAPO	123.4	123.5	121.3	121.1	121.4	120.3	118.5	128.4
mPL6	170.2	188.2	187.3	149.1	145.3	159.9	149.6	139.9
NTUPlace3	171.1	170.3	152.1	170.8	166.7	176.9	189.9	173.4
Rooster	136.6	163	156.5	147.4	152.8	169.4	162.7	174.2
FastPlace3	211.9	236.1	237.3	234.7	222.2	224.7	240.2	284.9
SimPL	124.7	118.8	117.8	116.2	115.1	114.8	114.1	119.5
SAPTgp + SAPTdp	100.4	100.1	100.1	100.1	100.1	100.1	100.1	100.1

Table 3.14: The peak weighted congestion (PWC) using the BFG-R [29] router and evaluation script from the DAC 2012 routability-driven placement contest on the MISPD datapath benchmark B variants on legalized placements. Routing resources are extracted from current 22nm technology node.

	ISPD 2011 Datapath Benchmark B: Average PWC							
Utilization	95	93	91	89	86	84	81	79
CAPO	237.4	222.4	222.5	217.4	222.3	220.1	224.3	219.6
mPL6	278.2	325.2	313.2	300	275	315.9	302.9	361.4
NTUPlace3	262.5	237.4	239.4	241.4	247.2	247.1	238.6	259.2
Rooster	256.2	224.1	217.3	215.1	231.5	221.8	228.7	225.2
FastPlace3	341.8	316.2	331.6	334.1	397.6	347	341.4	341.7
SimPL	207.8	211.3	205.2	205	212.5	209.3	204.6	209.2
SAPTgp + SAPTdp	151.9	150.7	149	148.5	147.9	148.1	149.2	150.1

Table 3.15: Total Overflow (TOF) using the ISPD 2011 contest router (coalesCgrip [74]) and the peak weighted congestion (PWC) using the DAC 2012 contest router (BFG-R [29]) for the hybrid designs C, D and E. Dragon results omitted as routing was unable to complete. Routing resources are extracted from current 22nm technology node.

	Hybrid C		Hybrid D		Hybrid E	
Routing Metrics	TOF	PWC	TOF	PWC	TOF	PWC
CAPO	1458	107.34	3024	110.46	26504	117.29
mPL6	1360	106.9	940	101.88	23838	117.02
NTUPlace3	626	103.66	558	95.92	32545	133.99
Rooster	48	100.38	1419	105.22	32157	122.32
FastPlace3	372	102.25	722	99.52	77774	165.88
SimPL	650	103.31	475	94.6	19713	118.73
SAPTgp + SAPTdp	534	103.49	322	93.21	11276	114.82

Table 3.16: Total Overflow (TOF) using the ISPD 2011 contest router (coalesCgrip [74]) and the peak weighted congestion (PWC) using the DAC 2012 contest router (BFG-R [29]) for the hybrid designs F, G and H. Dragon results omitted as routing was unable to complete. Routing resources are extracted from current 22nm technology node.

	Hybrid F		Hybrid G		Hybrid H	
Routing Metrics	TOF	PWC	TOF	PWC	TOF	PWC
CAPO	41520	114.73	131543	155.56	1186	115.52
mPL6	17801	111.17	45904	122.08	490	106.51
NTUPlace3	114843	141.76	42917	121.33	98	101.36
Rooster	61731	118.8	134434	149.26	918	111.49
FastPlace3	34189	118.21	33723	115.24	30	100.40
SimPL	29524	113.67	37884	120.33	100	101.11
SAPTgp + SAPTdp	28736	113.66	40275	119.12	28	100.34

state-of-the-art placers with the largest design, Hybrid F, taking under 71 seconds to place.

Table 3.17: Comparison of runtime on the hybrid designs.

Hybrid	C	D	E	F	G	H
CAPO	94.6	74.0	83.4	480.3	482.7	65.9
mPL6	48.5	32.4	36.2	161.7	148.5	35.9
NTUPlace3	13.0	30.0	70.0	278.0	269.0	54.0
Dragon	425.9	193.0	283.9	927.4	-	154.9
Rooster	98.8	84.1	101.1	560.7	455.1	80.9
FastPlace3	13.0	10.7	17.4	55.3	32.93	7.2
SimPL	9.2	12.6	27.1	59.2	40.4	9.2
SAPTgp + SAPTdp	15.9	16.7	38.2	70.9	43.1	10.8

3.8 Summary

Structured datapath layout is a long-standing challenge in physical design. Many attempts have been made in the last 40 years to close the quality gap between manual and automated placement of datapaths and other regular structures. Nevertheless, a common assumption still prevails among IC designers that circuits with high regularity require manual placement. A key observation in this chapter is that the primary optimization objective of modern state-of-the-art placement algorithms – HPWL can mislead placers on datapath-oriented designs. In particular, compressing placement of high-fanout nets to lower the overall HPWL can disrupt the regularity of placement and undermine its Steiner wirelength, which is known to better correlate with routed wirelength. Based on this observation, a unified framework is developed to enhance current random-logic placers to better handle designs

containing datapath logic seamlessly integrating alignment constraints into a state-of-the-art placement engine. Experimental results show at least a 28% improvement in total StWL compared with the state-of-the-art academic placers for the ISPD 2011 Datapath Benchmark Suite and a 5.8% average improvement in total StWL for industrial hybrid designs. Additionally, though not a congestion-aware placer, the techniques show that significant improvement in routability is achievable through datapath alignment.

Results presented in this chapter were impressive, but all datapath high-fanout nets required manual identification. In modern hybrid designs, these nets are not clearly defined. The next chapter presents techniques to automatically extract the datapath nets that need alighment.

Chapter 4

A High-Performance Placement Flow with Automatic Datapath Extraction and Evaluation through High-Dimentional Data Learning

The last chapter showed it was possible to alight datapath nets and cells on known instances. This chapter extends that to automatically identify the high fanout structured datapath nets within a design. The key discovery is that it is possible to characterize and extract specific structures that placers perform poorly on.

Structured or datapath extraction techniques in the past generally focused on functional or structural levels. Functional regularity extraction identifies logically equivalent subcircuits within a netlist that are then handled separately during placement. However, functional regularity alone does not imply placement will do a poor job. This has lead to the general industry practice of manually designing the datapath because of the possible significant timing and wire-length improvement possible. Yet, increasing design sizes and shortening turn-around-time demand a consolidated automated datapath extraction and placement framework.

In response to this, a new approach to extraction is proposed in this chapter. This chapter presents a high-performance placement flow with automatic datapath extraction and evaluation through high-dimensional data learning (PADE). Instead of searching for functional or structural regularity, the flow automatically extracts placement structures that are likely to be suboptimally placed.

PADE applies novel data learning techniques to train, predict, and evaluate potential datapaths using high-dimensional data such as netlist symmetrical structures, initial placement hints and relative area. Extracted datapaths are mapped to bit-stack structures that are aligned and simultaneously placed and with random logic. Results show at least 7% average total Half-Perimeter Wire Length (HPWL) and 12% Steiner Wire Length (StWL) improvements on industrial hybrid benchmarks and at least 2% average total HPWL and 3% StWL improvements on ISPD 2005 contest benchmarks. This is the first known attempt to link data learning, datapath extraction with evaluation, and placement and has the tremendous potential for pushing placement state-of-the-art for modern circuits which have datapath and random logics. This is an extension of the preliminary work presented in [88].

4.1 Introduction

Advancements in random logic placement have been impressive over the last few years with modern placers able to handle over one million placeable objects in minutes (e.g., [37]). Typically, these placers optimize the half-

perimeter wire length (HPWL) objective standardized by the 2005 ISPD Placement contests [92] and is a good indicator of placement quality for random logic designs [53]. Unlike random logic, datapath logic generally is characterized by a high degree of bit-wise parallelism [54] (often called bit-stack) that modern placers have shown to be suboptimal [92]. This is partly due to the inaccuracy of the HPWL model when compared to the Steiner wire length (StWL) [90].

Figure 4.1 shows a toy example where modern placers are not able to handle datapaths effectively. Figure 4.1(a) displays the datapath circuit, where the input and output pins are fixed. Cell 1 is an inverter driving four $NAND2$ gates and there are three bit-stacks corresponding to cells: $(\{2,3,4,5\}, \{6,7,8,9\}$, and $\{10,11,12,13\}$. For clarity, Figure 4.1(b-c) display fixed pin locations. Figure 4.1(b) displays the PADE placement solution solution. In this case, each bit-stack is tightly packed and aligned producing an StWL solution of 524. Figure 4.1(c) displays the placement solution from FastPlace3 [80] where the bit-stack is not carefully aligned producing StWL of 612. In fact, with only thirteen cells, the StWL solution in 4.1(c) is over 14% worse than that in 4.1(b). In designs where there are many embedded datapaths, extracting the datapath and placing them with random logics properly has the potential for significant improvement in the overall StWL.

Datapath extraction techniques in the past generally focused on functional or structural levels. Functional regularity extraction identifies logically equivalent subcircuits within a netlist that are then handled separately during placement. One example of this method is developed in [15] where a large

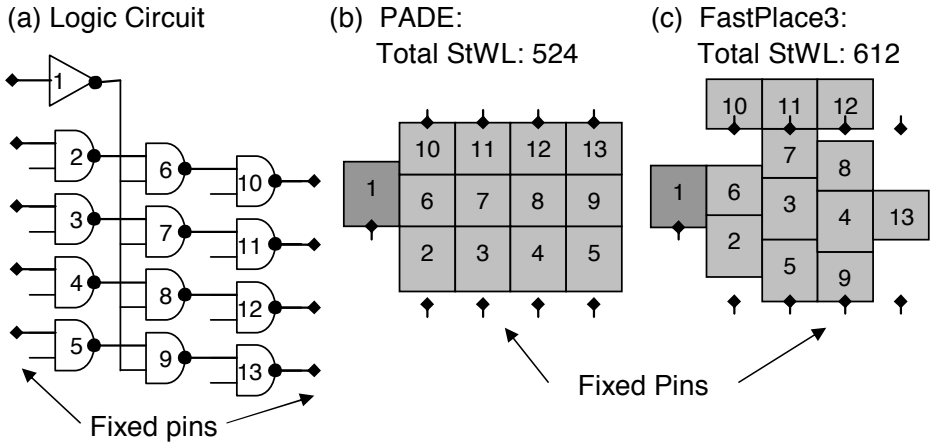


Figure 4.1: PADE placement example showing a 14% StWL improvement compared to FastPlace3 [80].

set of templates are generated and used to search for datapath logic before placement. Another example is the hash-based approach of [65]. Structural datapath extraction techniques have focused on developing a regularity metric to represent the datapath. In [54], the datapath extraction consists of a decomposition of the netlist into a set of stages and a set of slices with one cell occurring in exactly one stage set and one slice set.

The extraction algorithm expands in search-waves through the network using the regularity metric to determine the expansion direction. More recently, [55] developed a method for extracting structure within a design with the assumption that the placement distance between a pair of cells is related to the graph distance between them. Nets are weighted in a shortest path computation by assuming the distance between two cells is related to the degree of the net connecting them. Then, by extracting "corner" cells and fixing

them in place, the maximum distance of the other cells can be calculated.

These previous datapath extraction algorithms, which only use functional or structure information, are not effective for modern large-scale hybrid datapath/random logic circuits with many pre-placed IP blocks: (1) the placement blockage could force the bit-stacks to be placed away from adjacent logic causing wirelegnth degradation; (2) it may give out too many or the wrong bit-stacks which would also adversely affect the overall placement quality. Compounding the problem, as shown in [31], a dedicated datapath placer often overly constrains the random logic placer. These problems lead to the general industry practice of manually designing the datapath because of the possible significant timing and wire-length improvement by careful alignment and packing of the bit-stack. However, increasing design sizes and shortening turn-around-time demand a consolidated automated datapath extraction and placement framework.

This dissertation proposes PADE, a new placement flow with automatic datapath extraction which can handle large scale designs mixed with random and datapath circuits. PADE evaluates and ranks all the first-order, important data paths, and optimizes them along with general-purpose wirelength driven placement[1]. Once extraction is complete, PADE uses the SAPT [90] placer (which extends the simPL [37] global placer) to simultaneously place datapath and random logics.

[1]The name PADE is inspired by the famous Pade approximation which is widely used in model order reduction, as they share the same principle to extract the first-order effects.

The key contributions of this chapter include:

1. A novel high-dimensional data learning, extraction, and evaluation algorithm for datapath extraction is developed. It considers not only logic structures, but also placement hints from initial global placement results. The flow is generic and can be applied to other state-of-the-art placers.

2. An optimal algorithm for datapath cell selection (to guide data-path aware placement) using integer linear programming is designed.

3. Results demonstrate that PADE demonstrates significantly better results than previous state-of-the-art placers on both hybrid industrial designs which contain both random logics and datapaths, and even the ISPD 2005 placement benchmarks where structured datapath logics were not intended.

Section 4.2 outlines the overall flow and Section 4.3 details the high-dimensional extraction data. Section 4.4 describes the model training and cluster evaluation and Section 4.7 describes the binary integer programming datapath cell assignment technique. Experimental results are presented in Section 5.8 followed by conclusions in section 4.9.

4.2 Overall PADE Flow

Given a netlist $N = (V, E)$ with nodes V and nets E, placement obtains locations (x_i, y_i) for all movable nodes, such that the area of nodes within

the placement boundary does not exceed the area of cell sites in that region. With $\vec{x}, \vec{y} = \{x_i, y_i\}$, HPWL is defined as: $HPWL(\vec{x}, \vec{y}) = HPWL(\vec{x}) + HPWL(\vec{y})$ where $HPWL(\vec{x}) = \sum_{e \subset E}[MAXx_i - MINx_i]$. Modern placers often approximate HPWL by a differentiable function using the quadratic objective, defined as:

$$\Phi_G(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j}[(x_i - x_j)^2 + (y_i - y_j)^2] \tag{4.1}$$

From Equation (4.1), (x_i, y_i) represents the coordinates of cell i, and $w_{i,j}$ represents the weight between cells i and j. A datapath netlist with p bit-stacks, each bit-stack B_k $0 < k < P$ is a disjoint set of cells $B_k \subset V$, describing the bit-wise parallelism present in the netlist. Representing the datapath as a set of cells in this manner enables implicit StWL optimization through forced alignment as presented in [90].

PADE is a new placement flow extending the SAPT[90] framework with novel autmatic datapath extraction. The flow was built using the SimPL [37] force-directed global placer where $w_{i,j}$ is given by the Bound2Bound net model [76] and it utilizes a detailed placer similar to FastPlace3 [80].

Briefly, SAPT is a datapath aware placement flow requiring manual definition of both the bitstack and the datapath direction. For each bit-stack B_k, an alignment net is inserted, similar to a pseudo-net but remains persistent between placement iterations, connecting each cell in B_k. The alignment net is manipulated through the use of skewed weighting on $w_{i,j}$ and modified fixed-

point insertion making it possible to introduce an alignment constraint to a predefined group of cells within the linear solver during global placement.

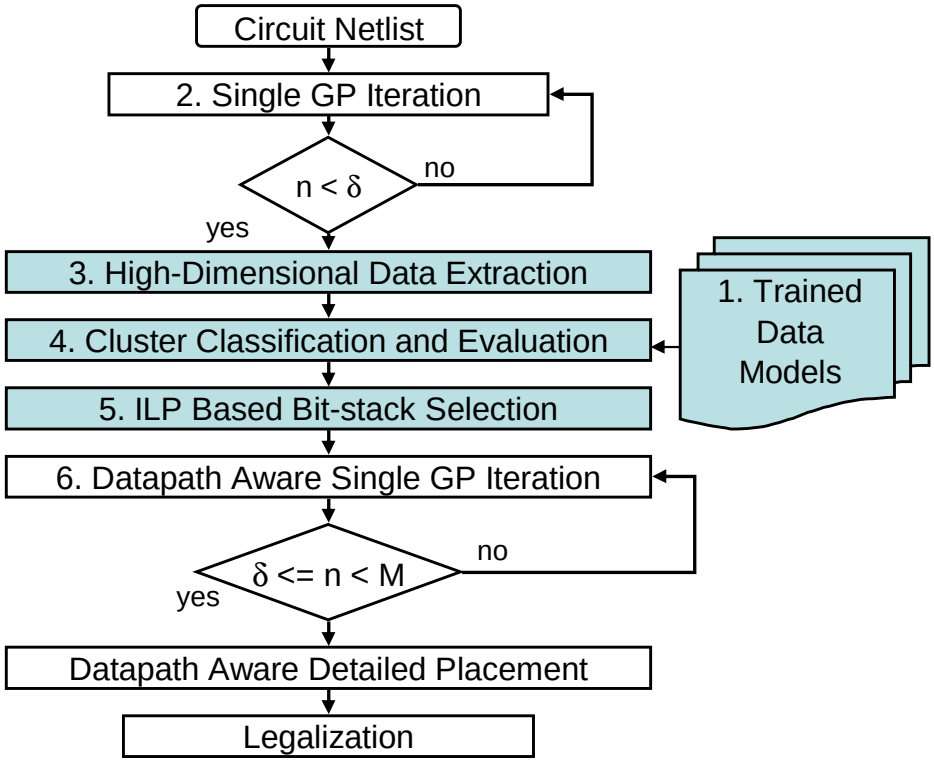


Figure 4.2: Overview of the PADE placement flow.

A gradually increasing application of this constraint aligns each group of datapath cells through consecutive global placement iterations, which enables a unified placement framework that simultaneously places datapath and random logic cells without over constraining the placer. Additionally, during detailed placement the placer maintains that alignment by constraining cell movements along the bit-stacks.

The overall PADE flow is shown in Figure 4.2 with the novel datapath training, extraction, evaluation and datapath bit-stack selection stages shaded. To properly handle the extraction of datapath structures, a compact and high performance knowledge base is proposed to identify datapath patterns from non-datapath logics and to evaluate the placement quality of these patterns.

Step 1., the knowledge base in Figure 4.2, is constructed via performing advanced data learning algorithms over a set of baseline design benchmarks after placement. By construction, it explores the placement database and captures the special characteristics that strongly correlate to datapath patterns, such as netlist connectivity automorphisms. Once these characteristics are captured and extracted, a complex decision diagram is built at a one time cost, which can later be applied to classify datapath netlists and non-datapath logics very rapidly. This knowledge base allows special treatment of the datapath logics in the placement stage without degrading the performance of the rest of the design. Additionally, these models are generic and can be applied to any circuit.

Step 2., a single global placement (GP) iteration, is one standard iteration of a force-directed placer that includes pseudo net insertion, linear system solver and fixed point generation. Let M, $0 \leq n \leq M$, be the upper bound on the global placement iterations and d an intermediate point at which time a prediction on the datapath will be made. Integrating the datapath extraction during global placement (GP) instead of before allows for enhanced prediction accuracy by taking into account physical characteristics in addition to netlist

regularity measures.

Step 3 extracts the high-dimensional features from the netlist and step 4 classifies and then evaluates the datapath candidates. For all identified datapath logics, step 5 uses an ILP formulation to map cells to a bit-stack and then inserts the alignment nets. Step 6 performs datapath aware global placement with bit-stacks aligned during the following global placement iterations. The flow completes with datapath aware detailed placement and legalization as described in [90].

4.3 High-Dimensional Extraction

In this chapter, both graph-based and physical features are analyzed and extracted from the netlist mapping a set of parameters most critical and sensitive to datapath logics. Effective features create differentiation between random and datapath logic allowing the patterns extracted on the training set to effectively classify datapath structures in new circuits and predict the direction of the datapath. The first step in this process is to generate candidate clusters of the original netlist in which to search for datapath structures.

4.3.1 Seed-Based Connectivity Clustering

The connectivity based clustering stage prepares the data to analyze and extract datapath structures from. The goal is to find clusters exhibiting the structure being sought. Extending the seed growth method proposed in [44], the clustering method creates k clusters. It maximizes the ratio of the

external to internal force of a cluster C_i, where C_i $0 \leq i < k$ indicates a group of vertices, while maintaining a maximum logic depth threshold. The external force is defined as the summation of the edge weights of nets with at least one vertex outside and one inside C_i and the internal force is defined as the summation of all internal cluster weight connections. The weight $w(u, j)$ is determined by the net model used, in this case a clique representation, where a connection c for a given edge e, $w(c) = w(e)/((|e| - 1)|e|)$. The connectivity between neighbor node u and cluster C_i is given by $conn(u, C_i) = \sum_{j \subset C_i} w(u, j)$ where suitable seed nodes are those with a large net degree.

In each subsequent pass, the neighbor node with the largest connectivity *conn* is added to the cluster C_i while keeping the internal force of the cluster as large as possible. Once a cluster's node cardinality reaches the threshold value or the entire netlist is clustered, the high-dimensional features described in the next subsection are extracted from each cluster C_i and then the cluster is classified as a datapath or random logic cluster.

4.3.2 Automorphism Feature Extraction

This section describes the graph features used to differenciate datapath and random logic. One of the fundamental observations in this chapter is that datapath logic contains a high degree of graph automorphism. An automorphism of a graph, a form of symmetry, preserves the edge - vertex connectivity

of the graph while mapping onto itself [2]. That is, an automorphism is a graph isomorphism from G to itself.

Definition 5. Automorphism: An automorphism of a graph $G = (V, E)$ is a permutation σ of the vertex set V, such that the pair of vertices (u, v) form an edge if and only if the pair $(\sigma(u), \sigma(v))$ also form an edge.

Automorphism Group: The set of automorphisms of a given graph forms the automorphism group of the graph and is denoted by $Aut(G)$. The set $S \subseteq Aut(G)$ of generators for $Aut(G)$ is a set whereby combining elements of S generates every non-identity permutation in $Aut(G)$.

Definition 6. Generator Set: A generator set of a group is a subset such that every element of the group can be expressed as the combination, under the group operation, of finitely many elements of the subset and their inverses.

As an example, Figure 4.3(a) displays a graph G with six labeled nodes and seven edges. The *automorphism feature* is represented with a seventeen parameter vector $(|Aut(G)|, \Delta(2:18),)$ where $|Aut(G)|$ is the cardinality of the automorphism group S_i for cluster C_i. The last sixteen parameters, $\Delta(2:18)$, are from the frequency table of the size of each automorphism.

As an example in Figure 4.3, the graph G has a total of four automorphisms and two generators ($|S| = 2$, with $|Aut(G)| = 4$). The first automorphism, $G(1, 2, 3, 4, 5, 6)$, corresponds to itself and three additional au-

[2] Assuming reader familiarity with graph automorphisms and permutations. Please refer to [32] for further details.

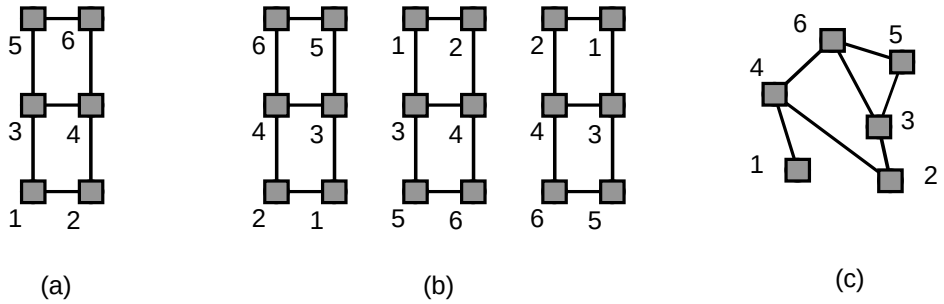


Figure 4.3: Graph automorphism example showing the original graph in (a) with each of the automorphisms in (b). A random netlist is shown in (c) with only trivial automorphisms.

tomorphisms $G(2,1,4,3,6,5)$ G flipped left-right, $G(5,6,3,4,1,2)$ G flipped up-down, and $G(6,5,4,3,2,1)$ G flipped left-right and up-down, displayed in Figure 4.3(b). The nontrivial generator set S of G is $(1,5)(2,6)$ and $(1,2)(3,4)(5,6)$. As this example shows, the symmetry of the graph along with the generator group provides possible bit-stack candidates including: $(1,2)$, $(5,6)$ or $(1,3,5)$, $(2,4,6)$.

Figure 4.3(c) displays a random logic netlist also with six nodes and seven edges. Unlike the clear symmetry present in Figure 4.3(a), Figure 4.3(c) contains no non-trivial automorphisms. In fact, this is a fundamental observation holding true for random logic netlists in general. Thus, the automorphism generators of structured logic appear very differently than the automorphism generators of random logic netlists enabling sufficient differentiation as a datapath feature.

4.3.3 Physical Aware Feature Extraction using Placement Hints

Graph automorphism features alone do not capture the physical nature of the placement problem, a fundamental shortcomming of prior extraction techniques. Global placement has merit in wirelength optimization, which shall be used for improved classification. Thus physical features extracted after the first few passes define the following attributes. Let a_i^c be the sum of the total cell area within cluster C_i, w_i^c be the bounding box width from the placement for C_i, h_i^c be the bounding box height and finally r_i^c be the ratio $a_i^c/(w_i^c + h_i^c)$. This physical information helps to characterize the amount of spreading and the initial cell locations for each C_i. Dense clusters indicate tightly packed logic and possibly the need for improved placement whereas sparse logic is generally less likely to improve from being passed to the datapath placer. In the next section, the training steps for building the model and the process to evaluate each cluster is described.

4.4 Datapath Model Training and Cluster Evaluation

To classify and evaluate the datapath patterns in each cluster, this work proposes to combine data learning algorithms Support Vector Machine (SVM) and Neural Network (NN) to build compact and run-time efficient models as shown in Figure 4.4. SVM calculates a hyperplane boundary with maximum separation margin in-between of datapath and non-datapath. Only the critical information on the separation boundaries is preserved (the support vectors SV). All SV's are involved in the decision (score) calculation.

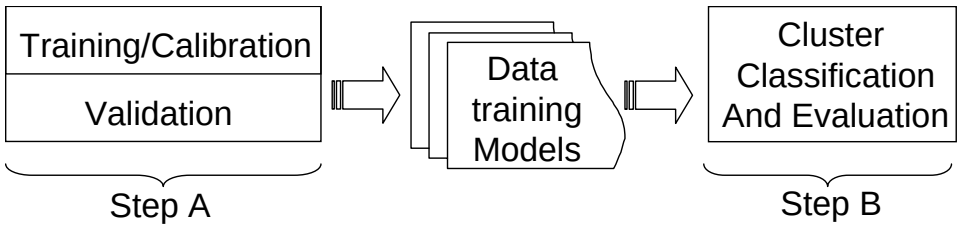


Figure 4.4: Major steps to build and apply the learning models

For better quality, a soft-error tolerant SVM and a special working set selection method [22] is combined. NN works through configuring complex networks of neurons to achieve a high dimensional decision diagram-like data structure given training samples and decision hints. A resilient backward propagation method is employed based on iterative sub-gradient updates. To quantify the learning performance, the following two types of accuracies are defined:

Definition 7. Datapath evaluation accuracy: the rate of correctly detected datapath (datapath-like) patterns over the total number of actual datapath structures.

Definition 8. Non-datapath evaluation accuracy: the rate of correctly detected non-datapath (e.g., random logic) patterns over the total number of non-datapath structures processed.

The optimization objective for both SVM and NN is to maximize the evaluation accuracies of datapath and non-datapath patterns, or equivalently, to minimize the mean square errors for both classes of pattern evaluation. This is achieved in two steps: Step A and B as shown in Figure 4.4.

4.5 NN Evaluation for Clusters

As briefly described in Algorithm 2, NN evaluates data samples by predicting a score for each high dimensional data V_p based on an established set of weights and biases assigned to certain neural network structure. The NN models are customized with single hidden layer of neurons, with transfer functions denoted as f_{hid}.

Require: Training vectors V_p's and training targets y_p's
  Scale each feature column of the training row vectors to [-1 +1]
  Divide training set into learning, cross-valid, cross-test sets
  Set converging speed parameters: $\eta_+ = 1.5$, $\eta_- = 0.5$, $\delta_{max} = 50$
  Initialize conditions for all variables to be updated
  while error target not met **do**
    for each V_p in the learning set **do**
      calculate gradients $\partial E^p/\partial \omega_{ij}$ and $\partial E^p/\partial \omega_{jk}$ for V_p
    end for
    calculate the average gradient value for each link as $\partial E^p/\partial \omega_{ij}$
    for all weights and biases **do**
      if $\partial E^p/\partial \omega_{ij}(t-1)$ * $\partial E^p/\partial \omega_{ij}(t) > 0$ **then**
        $sign = \eta_+$
      else if $\partial E^p/\partial \omega_{ij}(t-1)$ * $\partial E^p/\partial \omega_{ij}(t) < 0$ **then**
        $sign = \eta_-$
      else
        $sign = 1.0$
      end if
      $\delta(t) = \min(sign^*\delta(t-1), \delta_{max})$
      $\omega_{ij}(t)$ = -$\delta(t)^*sign\_func(\partial E^p/\partial \omega_{ij}(t))+\omega_{ij}(t-1)$
    end for
    update error for current epoch t
    break if(early stopping criteria met in valid and test sets)
  end while
  return NN model with ω_{ij},ω_{jk}

Algorithm 2: Pseudo-codes for training and calibrating NN

Inputs V_p to the NN are the extracted information (e.g., automorphism) and targets y_p are training knowledge (e.g., datapath information, placement hints). The variable p is used to represent the index of an input data, where $p = 1$ to N, V_p^i denotes the ith element of vector V_p, $i = 1$ to M, M is the total number information dimensions. The variable f_{in} and f_{out} are used to represent input and output layer transfer functions, and index i, j, k to indicate neuron indices in the input, hidden and output layer respectively. In particular, *sigmoid* functions are choosen for the hidden layer and a linear function for the output layer. Then the NN calibration process is formulated in Equation (4.2) to Equation (4.8) is formulated as follows:

$$objective : minimize\{\sum_{p=1}^{N} E^p\} \quad w.r.t \quad \omega_{ij}, \omega_{jk} \tag{4.2}$$

$$E^p = \frac{1}{2}[out_p - y_p]^2 \tag{4.3}$$

$$out_p = f_{out}\{\sum_j \omega_{jk} \cdot f_{hid}(\sum_i V_p^i \cdot \omega_{ij})\} \tag{4.4}$$

$$\frac{\partial E^p}{\partial \omega_{jk}} = (out_p - y_p) \cdot f_{hid}\{\sum_i V_p^i \cdot \omega_{ij}\} \tag{4.5}$$

$$\frac{\partial E^p}{\partial \omega_{ij}} = (out_p - y_p) \cdot \omega_{jk} \cdot V_p^i \cdot (1 + out_{hid}^j)(1 - out_{hid}^j) \tag{4.6}$$

$$f_{hid} = \frac{2}{(1+e^{-2x})} - 1, \qquad f_{in} = f_{out} = x \tag{4.7}$$

$$Est_{\tilde{p}} = \Delta\{f_{out}[\sum_{j} \omega_{jk} \cdot f_{hid}(\sum_{i} V_{\tilde{p}}^{i} \cdot \omega_{ij})]\} \tag{4.8}$$

As shown in Equation (4.2), the objective is set to the Summed Square Error (SSE) among all N input sample vectors. Such a minimization is achieved through iterative update of weight matrix $\overrightarrow{\omega}$ using modified resilient backward propagation method. Every iteration is called one epoch, defined as one complete representation of V_1 through V_N to NN. out_p is the evaluation result for each V_p data, while out_{hid}^{j} is the output from the jth node in the hidden neuron layer, both of out_p and out_{hid}^{j} are iteratively updated across epoches.

In particular there are 3 steps involved for datapath NN model training and calibration according to Algorithm 2, First, the input data set is normalized for each feature across the whole sample space. Second, the data set is divided into learning (80%), cross-validation (10%) and cross-testing (10%) subsets for the considerations of kernel establishment robustness, data overfitting prevention and proper early stopping criteria. In the third step, network output is adjusted incrementally with a step-wise update of network weight matrix towards minimal SSE value with parameters and gradient update procedures. With the calculations of the gradient values using Equation (4.4) to (4.7) and the arithmetic smoothing steps, the training is carried out iteratively until a certain error target is met.

Since NN training is only over a small presentative set of data, model validation is needed with more data before using to to evaluate new designs in the placement flow. The NN model is applied to a new set of patterns involving 300 datapath and about 60K non-datapath and the validation result is shown in Figure 4.5.

Based on the validation, both accuracies are balanced at the same time and 0.5 is selected as a separation threshold, giving 87.8% of datapath evaluation accuracy and 99.6% of non-datapath evaluation accuracy.

Require: V_p's from the *High-Dimensional Extraction*
Scale the features of V_p's correspondingly by the (*min, max*) values from Algorithm 2
Load *NN* model
Calculate Equation (4.8)
return A datapath estimate $Est_{\tilde{p}}$
Algorithm 3: Pseudo-codes for using NN

The NN training/calibration is performed multiple times and the best model is selected which gives the highest performance in validation. Once the model is ready, it is integrated into the placement flow using Algorithm 3, where the actual CPU run-time is negligible.

4.6 SVM Evaluation

The C-type SVM is given as follows in Equation (4.9) to Equation (4.13):

Require: training vectors V_p's and training targets y_p's
Scale each feature column of the training row vectors to [-1 +1]
Set control parameters: γ = -1/M, C = 1.5, stopping tolerance ϵ = 1e-3, min floating number τ = 1e-12
Initialize weight vector α and gradient vector G
while 1 **do**
  Working set (i, j pair) selection based on [22]
  if j == -1 **then**
    break
  end if
  Calculate $\eta_1 = \max(\ \tau,\ Q_{i,i} + Q_{j,j}$ - 2 $y_i\ y_j\ Q_{i,j})$
  Calculate $\eta_2 = y_j \cdot G_j$ - $y_i \cdot G_i$
  Update weight: ϖ_i += $y_i \cdot \eta_1$ / η_2, ϖ_j -= $y_j \cdot \eta_1$ / η_2
  $\varpi_i = slop\_func(\varpi_i)$,$\varpi_j = slop\_func(\varpi_j)$
  Update gradients for all $k = 1$ to M: G_k += $Q_{k,i}$ (ϖ_i - ϖ_i^{prev}) + $Q_{k,j}$ (ϖ_j - ϖ_j^{prev})
end while
Calculate ρ and *bias* values for prediction processes
return SVM model of non-zero elements of α and corresponding V_p's

Algorithm 4: Pseudo-codes for training and calibrating SVM

$$objective : minimize\{f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha\} \quad w.r.t \quad \alpha \tag{4.9}$$

$$subject \quad to : 0 \leq \alpha_i \leq C, i = 1, ..., N, \tag{4.10}$$

$$y^T \cdot \alpha = 0 \tag{4.11}$$

$$K(V_i, V_j) = exp\{\gamma \cdot \|V_i - V_j\|^2\} \tag{4.12}$$

$$Est_{\tilde{p}} = \Delta\{\sum_{i} \alpha_i y_i K(V_{\tilde{p}}, V_i) + bias\} \tag{4.13}$$

Require: V_p's from the *High-Dimensional Extraction*
Scale the features of V_p's correspondingly with the (*min, max*) values from Algorithm 4
Load SVM model
Calculate Equation (4.13)
return A datapath evaluation $Est_{\tilde{p}}$
Algorithm 5: Pseudo-codes for SVM Evaluation

Given samples V_i (i=1 to N) and the evaluation hints from known datapath information and placement results, first each data sample is labeled with a score y_i for the training process. In the formulation, e is a vector of all 1's and C is a pre-set upper bound to constrain feasible regions for SVM. Q is N by N positive semi-definite matrix defined as $Q_{ij} = y_i y_j K(V_i, V_j)$, where $K(V_i, V_j)$ is defined in Equation (4.12) as the kernel function. α is the N element weight vector for V_p's. Note α is generally sparse and the non-zero weights correspond to the final support vectors. Due to the fact that Q is usually dense and large, decomposition methods are usually used to solve the formulation iteratively rather than directly dealing with the quadratic Equation (4.9).

The training and calibration SVM models are achieved through performing Algorithm 4, which intakes data set V_p's and returns the supporting vectors and corresponding weight coefficients. There are 3 major steps involved: First, data set normalization for detection robustness; Second, high

order working set selection for enhanced detection accuracy particularly for the datapath placement requirements; Third, update weight and gradient vectors. The last 2 steps are carried out in an iterative manner until certain error target is met.

Similarly to NN, validation for SVM is performed after the training and calibration and show the accuracy as in Figure 4.6. Validation accuracy is observed around 88% and 99% for datapath and non-datapath, respectively, after selecting a separation threshold of -0.9.

The total model training/calibration time is about 3 minutes, and the validation time is negligible (less than a second). Algorithm 5 directly applies the SVM model to drive the placer. To compensate the disadvantages of NN and SVM, they are ensembled in the process of evaluating unknown patterns in the design, then the scores are used to drive the datapath placement flow.

4.6.1 Training, Calibration and Validation

In Step A, the data learning algorithms are applied over a relatively small set of design patterns with known datapath information under the guidance of placement as hints. Since they are built a priori at a one time cost, the CPU run-time penalty is negligible. There are 3 major procedures involved in this step: (1) training is the process where the learning algorithms optimize both datapath and non-datapath accuracies; (2) calibration process further improves the accuracies, e.g., via properly selecting the separation threshold in (1); (3) validation process is performed over a relatively large set of known

design patterns exclusive from (1) to assure the balance of learning accuracies between training data and unknown testing data, especially in Step B. These models can then be applied generically to any other designs to classify datapath clusters.

4.6.2 Cluster Classification and Evaluation

Once Step A is completed, in Step B the data learning models will be applied directly to classify and evaluate new unknown design patterns. As the new patterns go through the learning models, the evaluation scores could span within certain range for datapath and non-datapath patterns respectively for NN and SVM. This step evaluates a pattern to be datapath like if and only if both NN and SVM evaluation scores are above certain thresholds. This helps to systematically improve the datapath evaluation accuracy without noticeable penalty in non-datapath accuracy. Usually NN and SVM have similar performance for most of binary classifications, e.g., differentiating datapath-like and non-datapath patterns. In principle, SVM guarantees the global optimum but is sensitive to data noise. NN usually has good noise-robustness, however it takes more time in the training and calibration step to reach optimal or close-to-optimal. Each C_i identified as datapath logic is passed to the bit-stack assignment in the next section.

4.7 Bit-Stack Selection with ILP

Once a cluster has been classified as containing structured logic, step 5 from Figure 4.2 extracts the bit-stack structures from the logic clusters and passes those bit-stacks to the datapath placer. For each cluster C_i, a set of bit-stack candidates is chosen based on maximizing the total bit-stack count. This work uses the automorphism generators as the bit-stack candidates and adds in wire-length weighting to make the ILP formulation wire-length aware.

4.7.1 Bit-Stack Candidate List

Each generator set S_i, created during classification of each C_i, captures possible cell connections that can be used for a bit-stack assignment. Figure 4.3(a) provides an example for clarity. The generator group for the graph in Figure 4.3(a) is: $(1,2)(3,4),(5,6)$ and $(1,5)(2,6)$. Using this generator set, it is possible to create bit-stack candidates by grouping the tuple by index-0 and index-1 from each generator. For Figure 4.3(a), the bit-stack candidates would be:

$$b_0 = [1:3:5] \qquad b_2 = [1:2]$$

$$b_1 = [2:4:6] \qquad b_3 = [5:6]$$

Thus, with a set of bit-stack candidates, the goal is to maximize the number of bit-stacks within the partition while maintaining mutual exclusion among the cells. This constraint maintains the requirement that a particular cell can not be assigned to multiple bit-stacks.

4.7.2 ILP-based Bit-Stack Selection

The bit-stack candidate selection is optimally solved using integer linear programming (ILP). A binary vector η is maximized with the linear function $\Gamma^{\mathbf{T}}(\eta)$ subject to the non-overlap constraint. Assuming there are $i = 1...n$ bit-stack candidates, let η be a binary indicator variable such that:

$$\eta_i = \begin{cases} 1 & \text{if bit-stack candidate } B_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Let $\alpha_i = |B_i|$ and $\beta_i = w_i$ where w_i is equal to the Half-Perimeter Wire Length of the edges connected to each cell in bit-stack candidate B_i. The α_i term increases the value for larger bit-stack candidates (covering more cells) and the B_i term adds a penalty for larger wire-length. Then the objective function maximizes the number of bit-stack candidates selected as given in Equation (4.15).

$$\underset{\eta}{\text{maximize:}} \quad \Gamma = \sum_{i=0}^{n} [(\frac{\alpha_i}{\beta_i} * \eta_i)] \tag{4.15}$$

$$\begin{aligned} \text{subject to:} \quad & \eta_i + \eta_j \leq 1, && \forall\ i, j \iff \eta_i \cap \eta_j \\ & 0 \leq i < j < n && \forall\ i, j \\ & \eta_i \in (0, 1) && \forall\ i, j \end{aligned} \tag{4.16}$$

Equation (4.16) maintains the non-overlapping cell constraint $\eta_i \cap \eta_j = \emptyset$ between each bit-stack candidate. Though the general ILP problem is NP-Hard [5] and the solution time for the integer programming problem grows

exponentially (in the worst case) with the number of integer variables, in this case the run time is negligible for two reasons: (1) The number of bit-stack candidates n from a single C_i and the number of constraints is generally very low, with n often on the order of a few hundred because the size of C_i is bounded; (2) The ILP assignment only occurs when a cluster is classified as datapath logic meaning for the majority of the clusters, the ILP code does not run at all.

With the preceding steps from Figure 4.2, PADE is able to quickly extract and classify datapath structures then pass them to an ILP solver to generate the bit-stacks for the logic. By making the classification and bit-stack assignment aware of physical placement information from the global placer, significant improvement in overall wirelength is possible as will be shown in the next section.

4.8 Experimental Results

The first step, at a one-time cost, was training the high-dimensional models using known datapath pattern extracted from four baseline industrial hybrid circuits and the ISPD2011 Datapath Benchmark Suite [92]. Both datapath and random logic patterns were trained off the baseline circuits. Then the global placement flow was developed to extract and evaluate each C_i within the original netlist using the high-dimensional model. Clusters identified as containing datapath structures were mapped to bit-stacks using the automorphisms of the subcircuit and the ILP formulation. After the bit-stack was

defined, global placement continued through completion and then detailed placement and legalization ran. All numbers reported are total wirelength results for both datapath and random logics on legal placement solutions.

PADE was implemented in C++ with g++ 4.1.2 on top of the SAPT placement flow. Running PADE without any datapath awareness results in the same wirelength results reported for SimPL because SAPT was built on the SimPL framework. Benchmark runs were performed on an Intel Xeon CPU x5570 Linux workstation running at 2.93GHz using two CPU cores. This chapter compared PADE against six *untrained* industrial hybrid designs and additionally on the *untrained* ISPD 2005 benchmark suite [92]. For improved experimental control, all HPWL numbers and StWL estimates were generated using CoalesCgrip [74][3], every placer was run in *default mode*, and all placers were supplied a target density requirement of 1 as defined as in ISPD placement contests [92]. The tool bliss [32] was used to generate the automorphism groups for each cluster and GUROBI [56] for the lp solver. Wire-length results for the ISPD 2011 Datapath Benchmark circuits are not provided because they were used to train the high-dimensional models.

4.8.1 High Dimensional Learning Accuracies

Both SVM and NN algorithms were implemented and fine-tuned specifically for the evaluation of datapath patterns. Then both of their evaluation scores are combined for datapath extraction. The NN accuracy is shown in

[3]FastPlace3 [80] reports slightly lower HPWL than CoalesCgrip.

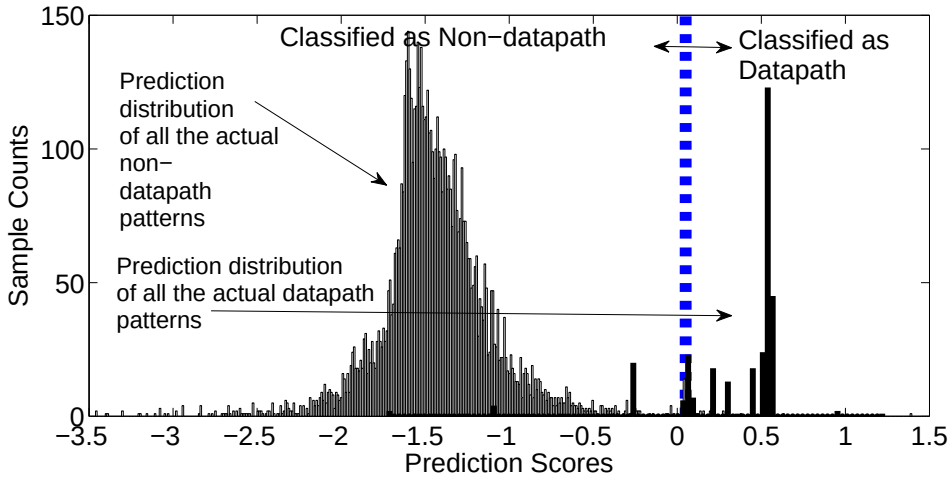


Figure 4.5: Validation accuracies of datapath and non-datapath by NN.

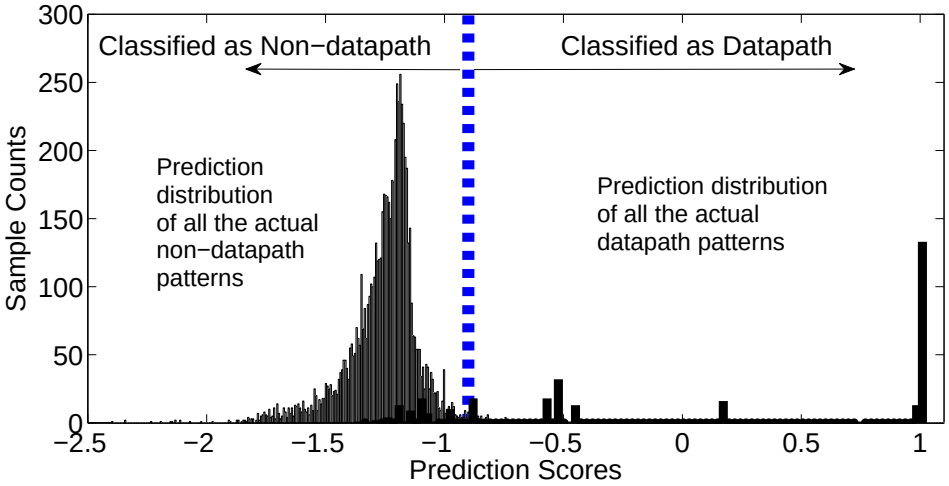


Figure 4.6: Validation accuracies of datapath and non-datapath by SVM.

Figure 4.5 and SVM accuracy is shown in Figure 4.6

A 2 class C-SVM algorithm is modified and configured at a one time training and calibration cost of around 3 minutes, involving: (1) training/calibrating of SVM models over some known structures with around 100 datapath and

10K non-datapath; (2) validation of the calibrated models over a relatively large set of known datapath/non-datapath structures beyond (1) with around 300 datapath and 60K non-datapath. Step(1) shows about 85% and 99.5% of datapath and non-datapath accuracy respectively, while Step(2) reaches 80% and 99% of datapath and non-datapath accuracy respectively. In the calibration and scoring process, a separation threshold of -0.9 is used for SVM models. A resilient backward propagation NN algorithm is fine-tuned within around 8 minutes using similar steps. It shows 90% and 99.9% of datapath and non-datapath accuracy in training, 87.8% and 99.6% in validation, with a separation threshold 0.05.

4.8.2 Wire Length Results

In the tables that follow, PADE refers to the proposed placement technique with automatic datapath extraction and evaluation. To compare the data learning and extraction effectiveness of PADE, *Logic Based Regularity Extraction* (LBRE) based on [40] was also implemented. Everything in LBRE is the same except the extraction and bit-stack assignment techniques. The logic based regularity extraction results are passed to the same datapath placer and compares the effectiveness of prior extraction techniques verses PADE. LBRE uses functional regularity to extract the datapath therefore can only be compared against the hybrid circuit designs because logical information is not provided in the ISPD 2005 benchmark circuits.

Wirelength results on six industrial hybrid circuits and the ISPD 2005

Table 4.1: Legal StWL (x10e6) comparison on industrial hybrid designs and the ISPD 2005 Placement Benchmarks [92]. StWL was computed using CoalesCgrip [74]. LBRE is blank for the ISPD 2005 suite because logic information is not provided for those circuits.

	[68] Total StWL	[6] Total StWL	[80] Total StWL	[12] Total StWL	[37] Total StWL	LBRE Total StWL	PADE Total StWL
Hybrid 1	3.12	2.89	2.61	2.67	2.75	2.89	**2.55**
Hybrid 2	2.51	2.17	2.2	2.2	2.2	2.18	**1.87**
Hybrid 3	2.75	2.41	2.28	2.25	2.25	2.26	**2.16**
Hybrid 4	3.57	4.01	3.59	3.36	3.3	3.49	**2.77**
Hybrid 5	12.9	14.41	13.27	12.3	12.22	12.22	**10.87**
Hybrid 6	9.06	10.29	9.04	10.69	7.92	8.21	**7.21**
Ave	**1.30**	**1.27**	**1.17**	**1.18**	**1.12**	**1.14**	**1.00**
Adaptec1	97.22	86.2	88.75	91.06	87.05	-	**85.12**
Adaptec2	114.54	100.64	104.03	99.06	102.13	-	**98.92**
Adaptec3	296.22	235.06	239.7	234.52	228.32	-	**222.08**
Adaptec4	257.47	208.85	215.02	211.86	201.82	-	**196.23**
Bigblue1	127.72	108.31	**105.24**	110.02	109.94	-	106.98
Bigblue2	189.6	174.69	178.44	175.27	168.65	-	**164.33**
Bigblue3	452.91	370.7	421.31	389.39	369.61	-	**361.96**
Bigblue4	1105.52	930.63	911.64	974.44	901.85	-	**883.82**
Ave	**1.22**	**1.04**	**1.07**	**1.06**	**1.03**	**-**	**1.00**

Placement Benchmarks are presented in Tables 4.1 and 4.2. Each of the hybrid designs are state-of-the-art circuits containing a mixture of random and datapath logic. Though the exact ratio of datapath to random logic is not known, generally the significant majority of the logic is random. As discussed in [90], HPWL to StWL correlation can be inadequate for datapath logic. Thus, both HPWL and StWL is reported with the best StWL result in bold. In every case, PADE obtains the best StWL results. In four of the six cases, PADE also outperforms all other placers in HPWL results.

Table 4.2: Legal HPWL (x10e6) comparison on industrial hybrid designs and the ISPD 2005 Placement Benchmarks [92]. HPWL was computed using CoalesCgrip [74]. LBRE is blank for the ISPD 2005 suite because logic information is not provided for those circuits.

	[68] Total HPWL	[6] Total HPWL	[80] Total HPWL	[12] Total HPWL	[37] Total HPWL	LBRE Total HPWL	PADE Total HPWL
Hybrid 1	2.39	2.27	2.06	2.04	2.19	2.32	2.05
Hybrid 2	1.72	1.47	1.39	1.38	1.39	1.39	1.37
Hybrid 3	2.68	1.89	1.81	1.77	1.77	1.79	1.71
Hybrid 4	2.66	3.18	2.91	2.35	2.69	2.79	2.36
Hybrid 5	11.36	12.76	10.88	10.59	10.57	10.56	9.71
Hybrid 6	7.66	9.04	7.75	9.04	6.64	6.9	6.24
Average	**1.25**	**1.23**	**1.11**	**1.09**	**1.07**	**1.10**	**1.00**
Adaptec1	88.14	77.58	79.88	81.82	78.15	-	76.83
Adaptec2	100.25	90.31	93.02	88.79	90.96	-	89.14
Adaptec3	276.8	215.88	219.78	214.83	208.81	-	205.32
Adaptec4	231.3	193.93	199.66	195.93	187.21	-	183.79
Bigblue1	110.92	97.1	94.37	98.41	98.64	-	95.86
Bigblue2	162.81	152.13	155.16	151.55	145.29	-	143.18
Bigblue3	405.4	342.5	392.72	360.66	341.55	-	341.72
Bigblue4	1016.19	831.34	816.14	866.43	804.22	-	796.18
Average	**1.21**	**1.03**	**1.06**	**1.05**	**1.02**	**-**	**1.00**

The purpose of running on the ISPD 2005 placement benchmark suite is to show that the methods herein are capable of high placement quality on both random and datapath logics. Surprisingly, some datapath structure was found and on average PADE improves the HPWL 2% and StWL by 3% compared to prior academic placers. As Tables 4.1 and 4.2 show, PADE produced the best HWPL and StWL results for seven of the eight benchmarks. One notable placer missing from the comparisons is the structure aware Beacon placer

[55]. Though requested, currently the placer does not work with mixed-size placement and thus direct comparison is not possible.

4.8.3 Runtime Comparisons

Table 4.3 compares the runtime of PADE against other state-of-the-art placers. For the hybrid and ISPD 2005 Benchmark circuits, FastPlace3.1 ran the fastest of all placers. For the hybrid circuits, PADE was only 19% slower than FastPlace3.1 and for the ISPD 2005 benchmarks, PADE was 32% slower than FastPlace3.1. Overall, PADE significantly outperforms CAPO10.2, mPL6, and NTUPlace3 showing speedups of 7.28x, 3.26x and 1.74x respectively on the ISPD 2005 Benchmarks.

Though PADE is not the fastest, there is clearly wirelength benefit on hybrid design styles and it is possible to parallelize the clustering, evaluation and bit-stack assignment stages of the flow. Doing so would reduce runtimes to be similar with the other state-of-the-art placement algorithms.

4.9 Summary

This chapter presented a new placement flow PADE with automatic datapath extraction and evaluation through high-dimensional data learning using both logical and physical information. PADE has demonstrated 7% improvements in HPWL and 12% improvements in StWL for a set of industrial hybrid circuits compared to prior placers. Even for the ISPD 2005 benchmark circuits, PADE produces 2% average improvements for HPWL and 3% im-

Table 4.3: The total runtime comparisons (sec). Runtimes on the ISPD 2005 benchmarks on LBRE are left blank because logical information is not provided by the ISPD 2005 benchmarks. (hd = hybrid, ad = adaptec, bb = bigblue, FP3.1 = FastPlace3.1)

	[68]	[6]	[80]	[12]	[37]	LBRE	PADE
Hybrid 1	7.4	1.9	1.3	1.5	1.2	2.1	1.1
Hybrid 2	8.1	2.4	1.4	1.7	2.0	2.7	1.7
Hybrid 3	8.8	4.5	1.7	2.7	2.2	4.2	1.8
Hybrid 4	8.6	4.1	2.2	3.2	2.2	6.3	2.3
Hybrid 5	25.2	10.7	6.4	9.8	5.8	12.8	7.4
Hybrid 6	45.4	22.8	4.8	9.3	4.1	14.2	7.1
Ave	**4.99**	**2.01**	**0.81**	**1.31**	**0.97**	**2.05**	**1.00**
Adaptec1	35.7	18.3	4.7	10.0	4.2	-	5.3
Adaptec2	42.8	19.9	2.2	9.2	4.4	-	5.6
Adaptec3	111.9	60.3	4.4	18.6	10.7	-	12.9
Adaptec4	110.0	58.5	9.0	19.5	18.1	-	21.4
Bigblue1	56.6	21.8	5.4	16.2	4.5	-	5.5
Bigblue2	107.6	64.0	9.6	32.1	17.3	-	20.4
Bigblue3	286.0	88.4	28.3	62.5	34.8	-	40.6
Bigblue4	543.4	172.8	58.1	141.8	62.3	-	73.0
Ave	**7.28**	**3.26**	**0.68**	**1.74**	**0.83**	**-**	**1.00**

provement in StWL over prior placers. The next chapter examines another form of structured placement by observing that clock trees are another form of high fanount net that are often suboptimally designed.

Chapter 5

Clock Power Minimization using Structured Latch Templates and Decision Tree Induction

This chapter explores the fact that local clock trees are another form of high fanout net. As shown in Chapter 3, when placing cells connected to high fanout nets, these cells are clustered together. This phenomenon also occurs during clock tree synthesis. By specifically optimizing the placement of the memory elements connected to the local clock tree, this chapter shows that it is possible to save significant improve the local clock tree.

Predicted for many years, power constraints have throttled performance scaling once experienced in multi-Ghz microprocessor design. These constraints now relegate process enhancements to minor speedups with little change expected in the near future. This necessitates costly power savings techniques such as multiple supply voltage islands, multiple threshold voltages, and aggressive power gating techniques. In spite of these efforts, power persists as the greatest challenge to both modern multi-GHz designs and low-power SoCs in nanometer CMOS technologies. Complicating the issue is the increasing on-chip variation (OCV), which produces a significant drop in yield [27][62] resulting in stricter design guide rules. These effects are particularly

poignant for clock design. Clock power often contributes between 40% and 50% [59][96][94] of total CPU power and any improvement results in meaningful overall chip power savings. Though clock design has been heavily researched, it is still a challenging and critical aspect of physical design as shown by the recent clock synthesis contest [78]. Compounding the power problem, skew requirements of multi-Ghz design has necessitated the need for a hybrid clock routing methodology where a low-skew global clock mesh overlays the entire die area followed by locally buffered clock trees [96] [94][98] often described as multi-source clock tree synthesis (MSCTS). This is an extension of the preliminary work presented in [93].

5.1 Introduction

An example of this methodology is shown in Figure 5.1. Design rules enforce strict skew constraints on the global clock mesh, which is located on upper metal layers and shown in green. The local clock buffer (LCB) connects to the global clock mesh at specific locations providing latch placement flexibility to the physical design (PD) automation tools.

This two-tiered approach came about for a number of reasons. First, though clock trees are lower power than clock meshes, trees do not offer low enough skew for multi-Ghz designs. Second, clock meshes are very power hungry and local clock trees (LCT)s offer a significant power savings. Third, LCTs reduce the local wire routing demands for routing the clock compared to full meshes at lower level metal layers. This methodology still faces challenges from

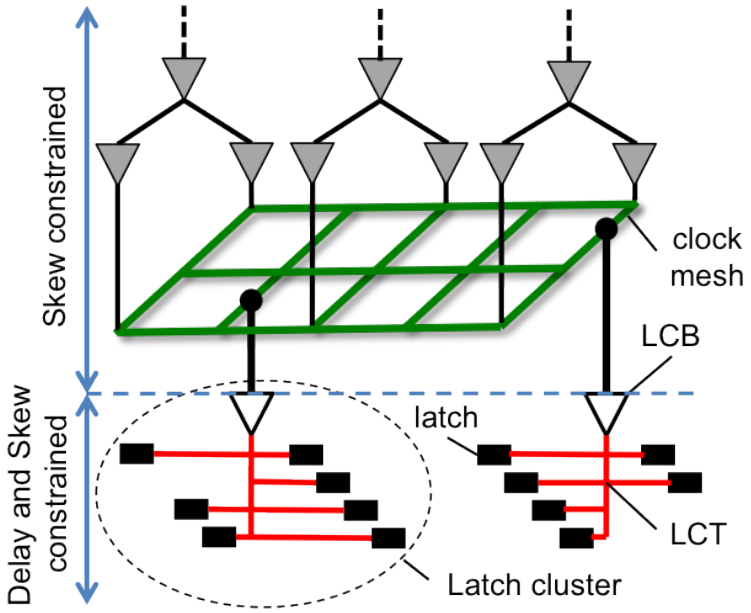


Figure 5.1: State-of-the-art high performance clock mesh methodology. The global clock grid uses a clock mesh with tight skew requirements. Local clock buffers (LCB)s connect to the grid and drive local clock trees (LCT)s to each individual latch. Design guide rules maintain strict skew and nominal delay constraints for each LCT.

increases in process variation and tightening design and OCV constraints making it difficult to generate correct-by-construction LCTs. Additionally, within an individual design, there could be thousands of these LCTs making accurate design time modeling and optimization difficult because of the runtime impact. In practice, overly pessimistic constraints are often applied to minimize electrical violations on the LCT, which results in extra timing closure cycles, significant redesign, or even engineering change orders (ECO)'s.

The key contributions of this work are as follows:

1. A methodology for significant power reduction is proposed by generating optimized latch cluster placement templates once for each technology library.

2. A correct-by-construction optimized latch cluster placement template flow is developed to meet electrical constraints reducing design time.

3. A framework is developed for reducing the required cardinality of the structured templates through set-theoretic annotation.

4. A machine learning based decision tree induction model with a novel distance metric is trained to quickly select the correct optimized template within the PD flow.

Section 5.2 outlines the background and presents a motivating example displaying the significant capacitance reduction possible through structured latch cluster placement templates. Section 5.3 presents the proposed overall template development flow and Section 5.4 details a genetic latch placement algorithm for identifying optimized placement solutions. Structured template generation and redundancy removal is proposed in Section 5.5 and the decision tree classification with novel distance metric is described in Section 5.6. Section 5.7 illustrates how the templates integrate into a modern physical design flow and lastly, experimental results are presented in Section 5.8.

5.2 Background and Motivation

Clock skew is the fundamental metric for evaluating clock performance. Optimizing the clock tree in the presence of this constraint has been approached from many aspects including clock gating, multiple clock domains, reducing local clock buffers (LCB) and register (latch) placement. Modern System-on-a-chip (SoC) and multi-Ghz designs utilize many, if not all, of these techniques to build robust low skew clock networks [27][9][10][97]. Recently, it has been shown that modifying the latch placement locations is an effective approach producing significant reduction in overall local clock tree (LCT) capacitance when compared to unconstrained placement [27][62][30][28]. By minimizing the clock tree capacitance in this manner, clock power is directly reduced.

There are three prior latch placement modification techniques, latch shifting, latch clustering, and latch banking. *Latch shifting* is the least disruptive approach where [30] showed that incremental shifts in latch placement toward preferred locations resulted in smaller clock trees. Though latch shifting is less disruptive, the reduction in LCT capacitance is also limited. *Latch clustering* is the most common LCT reduction approach when modifying latch placement. The work in [62] showed that clustering latches around LCB's significantly improved latch power and helped to meet design rule requirements. Clustering the latches around the LCB in this manner reduced LCT capacitance up to 50% when compared to unconstrained placement. "Length-constrained latch clustering" [59] generalized this concept, where a maximum

latch displacement parameter provided a tunable trade-off in LCB numbers versus layout disruption. *Latch banking* is the third approach. By automatically placing registers into fixed "banks" it is possible to reduce both clock power and skew [28]. Relative placement constraints are used to implement this approach but it is often not flexible enough to deal well with industrial challenges such as fixed obstacles and congestion [27]. Additionally, the LCT, though simplified, must still be implemented at runtime.

For all approaches, modifying the placement location of latches causes a timing degradation [59] because the clock optimization (clock opt) stage occurs after initial timing corrections have occurred. Banking is the most disruptive because it applies a fixed placement constraint without giving flexibility to the PD tool to select a more optimal structure. In spite of the disruptive nature, clustering has been widely adopted as a balanced approach because of the significant power savings. As such, modern PD flows have enhanced post clock opt steps to address the performance degradation [59][60] [90] [48]. The result is that PD flows are robust enough to overcome disruptions caused by the modified latch placement producing significant power savings with little delay impact. Because of this, latch clustering has become the de facto methodology for multi-Ghz designs.

Figure 5.2 shows snapshot of a multi-Ghz design employing the conventional clustering approach. As can be seen, small groups of latches are tightly clustered around an LCB significantly reducing the total LCT length resulting in direct power savings. With local *clock trees contributing between 20-30%*

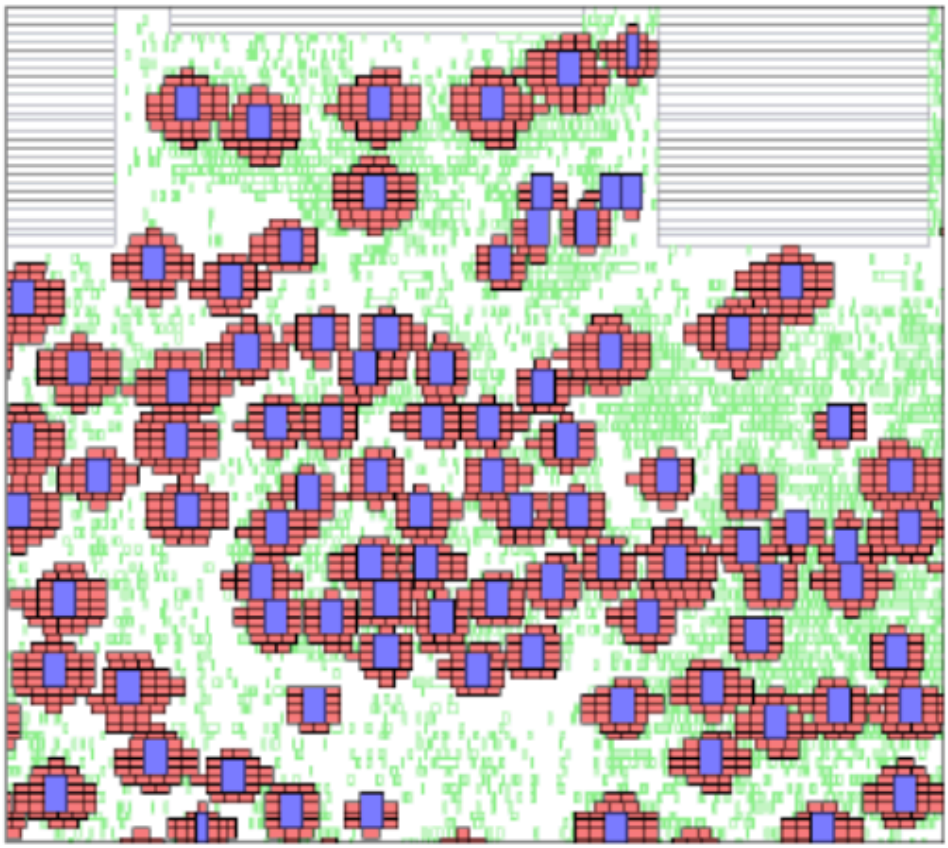


Figure 5.2: Multi-Ghz design showing conventional "clustered" latches. Red cells are latches and blue cells are LCBs. One-time computation of optimized templates for a technology library make it possible to significantly decrease local clock tree (LCT) capacitance resulting in reduced total power.

of the total dynamic power [96][94][41], even small improvements in the LCT equate to large overall power savings.

5.2.1 Conventional verses Structured Latch Clusters

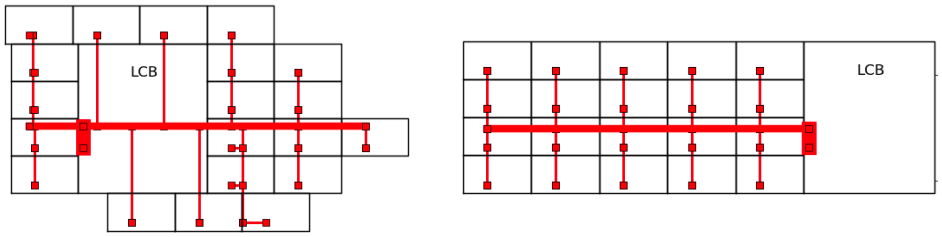
Extending the concept of conventional latch placement approaches, this work generates a set of optimized placement templates called structured latch

clusters one time for the technology library. Generated *a priori*, the templates specifically reduce clock tree capacitance on the LCB while meeting the maximum skew constraint. This provides the flexibility of the clustered approach with the power savings of the banked approach. In this work, as in the 2010 ISPD Clock Contest [78], the clock tree capacitance is the primary evaluation metric for comparing multiple latch cluster placement solutions because, as Equation (5.1) shows, reducing the C_{load} of the LCT directly reduces overall dynamic power. In addition to the reduction in capacitance, the placement templates are correct-by-construction resulting in reduced design iterations after clock optimization.

Figure 5.3 shows an example of the potential impact of structured latch placement verses a common tightly clustered solution. First, with only 20 latches, the structured latch solution reduces the total capacitance of the LCT by 33%. Second, even though the skew on 5.3(a) is better, both (a) and (b) meet maximum skew and delay constraints making both valid solutions. Third, 5.3(b) requires fewer local clock routing resources than (a). Fourth, current placement legalization techniques are much better suited at effectively legalizing solution (b) than solution (a) resulting in improved overall quality of results. Though not shown, the clock routing solution for conventional clustered approaches is also often more complex than a structured template. In the example presented, a single trunk route was required to meet the slew and skew constraints for both approaches. However, often that is not the case for conventional approaches. In cases where a horizontal and vertical trunk

route is required, the capacitive savings of a structured solution can be much larger.

$$P_{avg} = \frac{1}{T} \int_0^T v(t) * i(t)dt = C_{load} * V_{DD}^2 * f \tag{5.1}$$



(a) (b)

Figure 5.3: Placement of 20 latches around an LCB where (a) is a conventional clustered solution with latches pulled close to the LCB producing lower skew and (b) is the proposed structured latch placement solution with higher skew but still meeting design requirements. With only 20 latches, (b) reduces total capacitance on this LCT by 33%. The red wires show the LCT routing solution.

A *key fundamental observation of this work* is that current physical design flows are already producing packed latch cluster placements. Instead of calculating placement solutions during runtime, this work is identifying more optimal solutions *a priori* for the technology library and providing the physical design flow a runtime choice in which template to choose from. Additionally, by generating a wide range of placement templates, the PD flow has the flexibility to choose the correct placement template as opposed to a rigid latch banking approach.

5.2.2 Problem Complexity

Simply building a library of all possible structured template solutions is not practical. Modern microprocessor designs often contain more than *ten million* latches. With average latch cluster sizes between 20 and 30, that equates to over three hundred thousand local latch clusters on the microprocessor. However, any individual cluster can range between just a few or more than sixty. Additionally, twenty or more clock domains are common each with a unique set of latches. Within an individual clock domain, the standard cell library often contains latches with multiple drive strengths each resulting in different placement footprints. To optimize power, multiple LCB sizes are also needed each with different requirements for skew, slew and placement footprint. Complicating the issue, multiple placement footprints for the latch cluster itself are required to balance tradeoffs between capacitance, skew, LCB count, placement density, fixed blockages and local routing congestion. For example, if there is significant horizontal congestion, selecting a latch cluster that has minimal horizontal routing constraints is beneficial. Combined, these requirements contribute to over one hundred thousand possible input combinations, each with a potentially different structured latch cluster solution. Additionally, latch clusters commonly contain multiple latch sizes, pushing the number of possible solutions into the millions.

Clearly selecting between millions of placement templates during run time is not practical. Three key components are required to make this tractable: first, a systematic framework to reduce the cardinality of the optimized tem-

plate set, second, a generalized method for selecting the correct template, based on an unknown input combination, third, designer control because manual tuning is still a significant portion of the timing closure flow.

5.2.3 Benefits of "a priori" Optimized Placement Templates

In spite of these challenges, the proposed methodology offers five key benefits when compared against prior approaches.

1. Capacitance reduction, significantly reducing clock power, is possible through the use of optimized placement templates when compared to conventional approaches. Because of *a priori* generation, techniques to optimize the templates can sustain much longer runtimes.

2. Correct-by-construction placement solutions are possible with optimized placement templates unlike conventional approaches. This means they are almost guaranteed to legalize, to be overlap free, and meet design guide electrical and skew requirements.

3. Legalization of conventional clustered solutions can be challenging because the footprint is not guaranteed. This often causes many placement overlaps requiring further perturbations in the design closure flow. The placement footprint of the structured template solution however is very compact and known *a priori* reducing the number of timing and placement disruptions.

4. Routing aware latch placement is possible with optimized placement templates. Conventional approaches are unaware of local clock routing topology often causing localized pin accessibility issues and potentially severe congestion. With the use of structured latch cluster templates, placement solutions can be selected that mitigate congestion.

5. Runtime is a precious commodity during state-of-the-art physical design flows. By generating the optimized placement templates *a priori*, once for each technology library, compute time can be freed for other optimizations.

This work proposes a scalable solution to this problem by first generating a large set of initial placement templates. Then significantly reduces the cardinality of the solutions by using the set theoretic difference to remove redundancy. Finally, using a machine learning technique called decision tree induction, a model is developed, using a novel similarity metric for quickly selecting the correct template during design automation. Additionally, the proposed framework lends itself to be easily communicable to designers in a manner that makes it easy to interpret. As results will demonstrate in Section 5.8, significant capacitance reduction on the LCT is possible through the use of the proposed approach. The next section outlines the overall flow of the proposed template design methodology.

5.3 Structured Template Development Flow

This work proposes a fundamental shift in latch placement methodology by proactively identifying optimized placement configurations with significantly lower capacitance one time per technology library. The proposed *a priori* template development flow is segmented into three key stages, *Template Development*, *Decision Tree Classification*, and *Classification Model Deployment* as shown in Figure 5.4. *Template development*, (Section 5.4) is comprised of two stages, placement search and template generation. The placement search stage uses a genetic algorithm to search for a latch cluster placement solution that is as close to optimal as possible. Then, the template development section, (Section 5.5), reduces the number of placement solutions by calculating the set-theoretic difference of the redundant templates and removing them. *Decision tree classification*, (Section 5.6), is a machine learning based classifier comprised of two stages as well. The first stage trains a supervised decision tree to map the input set to the generated templates from the prior stage. The second stage validates the decision tree on unseen patterns to verify generalization of the model. Finally, *Classification model deployment* provides the learned models for deployment during clock optimization of a normal physical design flow. All of this is generated at a one time cost resulting in little to no runtime impact from the proposed approach. The next section provides the details of the template development stage.

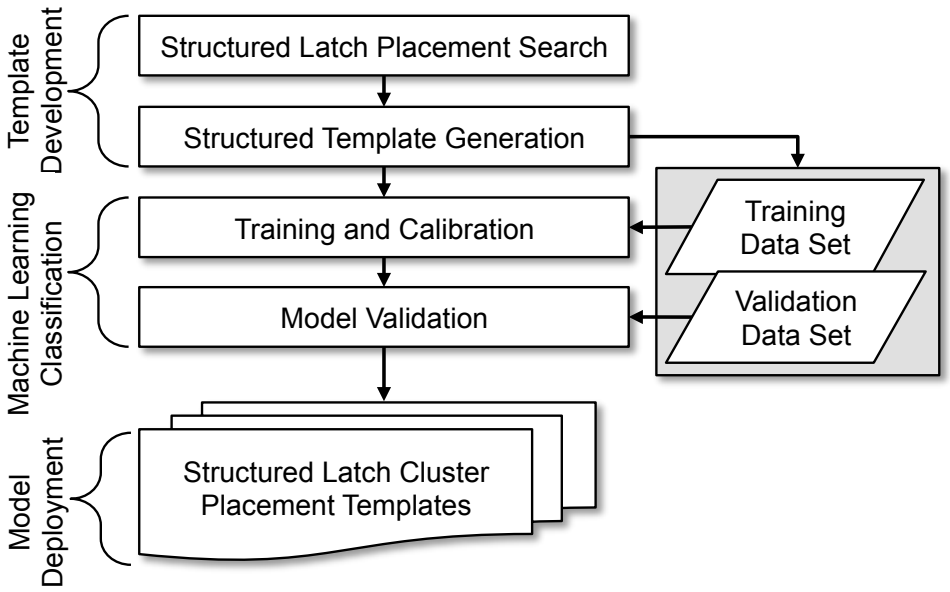


Figure 5.4: Overall optimized structured template development flow design flow illustrating each stage of the proposed flow. Each is generated *a priori* per technology library at a one time cost. resulting in little runtime impact from the proposed approach.

5.4 Structured Register Placement Search

This section develops a latch cluster placement approach to implicitly reduce clock power through routed capacitance reduction while meeting design guide rules. Let F be the set of possible templates. An input vector E that results in a template $t \in F$ consists of a number of components. First is the latch cluster G consisting of two types of placeable objects, a set of latches L and an LCB, $(L, LCB) \in G$, where each latch and the LCB have a height h and width w. Second *bounding box ratio range* $\beta_{max} - \beta_{min}$ is defined as the minimum and maximum value of the placement width of the template divided by the placement height. Providing a ratio constraint gives flexibility to select a "shape". Third, the *density range* $\delta_{max} - \delta_{min}$ is defined as the structured template density. A high-density template, as displayed in Figure 5.3(a), often has slightly increased capacitance requirements but is generally easier to legalize than a template that is less dense. Therefore, templates with a range of density requirements are generated. Fourth, a maximum skew constraint d_{max} for the latch cluster is provided as input. Finally, a minimum and maximum delay constraint $r_{max} - r_{min}$ is defined as an input requirement. In this work, exhaustive combinations of these input requirements are passed to the template development stage.

Given an input vector E containing G, $\beta_{min,max}$, $\delta_{min,max}$, $r_{min,max}$, and d_{max}, placement generates locations (x_i, y_i) for all placeable objects in G such that routed capacitance C_r of the local clock tree (LCT) is minimized. This is shown in Equation (5.2).

$$\begin{aligned} \text{minimize:} \quad & C_r \\ \text{subject to:} \quad & d < d_{max} \\ & r_{min} \leq r < r_{max} \\ & \beta_{min} \leq \beta < \beta_{max} \\ & \delta_{min} \leq \delta < \delta_{max} \end{aligned} \tag{5.2}$$

A genetic search (GA) is proposed to generate the (x_i, y_i) placement locations for a optimized structured template solution. Briefly, a generic GA search begins with an initial population and applies operators to create new populations to successive generations. Reproduction is the first operator where chromosomes are copied to the next generation with some probability based on a fitness criterion. The second operator is crossover where weighted random pairs of chromosomes are mated creating new chromosomes. Mutation is the third operator occasionally altering a portion of the chromosome. The crossover is the most critical component for an effective GA and mutation periodically diversifies the search space. Further details of the GA are discussed in the following subsections with the overall algorithm presented in Section 5.4.4

5.4.1 Ordered Candidate Representation

An important requirement for utilizing a GA is the ability to represent the solution space as a set of objects, referred to as the genetic coding. In

this case, each structured latch cluster (SLC) G is represented as a strictly monotonically increasing register sequence a_i, where $1 \leq i \leq |G|$. Each register $a_i \in G$ represents the tuple (x_i, y_j) where x_i and y_i are the x-axis and y-axis respectively of the placement coordinate locations for register a_i within the placement region. Strict monotonicity is maintained $\forall a_i \in G$ given by Equation (5.3). This representation allows encoding each G in such a way that crossovers of feasible chromosomes result in feasible chromosomes.

$$\forall a_i \in G \begin{cases} a_i(y) \leq a_{i+1}(y) \\ a_i(x) < a_{i+1}(x) \end{cases} \iff a_i(y) = a_{i+1}(y) \tag{5.3}$$

5.4.2 Fitness Function

A second important requirement for utilizing a GA is an effective fitness function to score each solution. The fitness function reflects the goal to minimize the total routed capacitance C_r of G. The first component in the proposed fitness function, shown in Equation (5.4), is the total routed capacitance C_r of the structured latch cluster (SLC). The second is the clock skew, modeled as the cost function shown in Equation (5.5) where d is the skew of the placement solution. Figure 5.5 plots this cost function η_d.

$$f(x) = \frac{1}{C_r(\eta_d + 1)} \tag{5.4}$$

$$\eta_d = \begin{cases} 0 & d_{slc} < d_{max} \\ \lambda \cdot (1 + d_{slc} - d_{max}) & d_{slc} \geq d_{max} \end{cases} \tag{5.5}$$

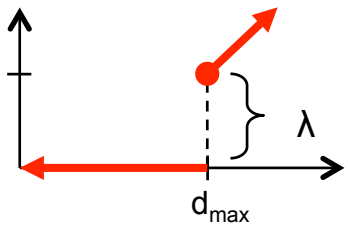


Figure 5.5: Skew constraint modeled as a cost function. η_d gives preference to placement solutions meeting the skew constraint. For this work, λ is kept default across all experiments.

Maintaining the β, δ and r constraints is accomplished with an infinite cost function. Namely, chromosomes in violation of those constraints are immediately discarded. An infinite cost function for skew was evaluated, but proved suboptimal compared to the proposed linear function. In the infinite case, crossing the skew boundary guaranteed the candidate solution was discarded immediately and did not mate into future valid chromosomes.

5.4.3 Order Crossover for Structured Latch Clusters

Crossover is the operator applied to a pair of parent chromosomes selected from the best solutions in the prior population. Creating new chromosomes from current ones (the crossover technique) effectively searches the solution space in a GA. In this work, that means selecting two initial parent placement solutions and generating a new placement solution based on

portions of the parents. A common crossover technique [69],[16], the ordered crossover, quickly and effectively generates new candidate solutions making it an attractive option. For parent templates a_1, a_2, two child templates b_1, b_2 are generated as copies of the parents. Next, two positions are selected at random in the interval $[1, |a|]$ and a portion of a_1 up to the first position is mapped to b_2 and a portion of a_2 from the second position is mapped onto b_1. From the respective position on, the child template is filled with the opposite parent chromosome. This approach maintains the order from the parents in positions that were mapped over.

5.4.4 Overall Placement Algorithm

The overall GA algorithm is presented in Algorithm 6 and begins by randomly initializing a population. Correct-by-construction templates require legalized SLC placement solutions so the next step is overlap removal (step 2) using standard legalization techniques from [80]. Step 3, register collapsing, removes whitespace between registers within a row by collapsing them toward the LCB.

Fitness calculation is next for the entire population with the evaluation metric defined in Equation (5.5). Reproduction chooses a weighted selection of the best chromosomes in the prior population. Step 6 mates the two selected chromosomes using the ordered multiple crossover technique and mutation randomly shifts a register to a new location. Steps 8 and 9 remove overlaps in the new population and collapse the registers toward the LCB. Finally, the

Input: G, $\beta_{min,max}$, $\delta_{min,max}$, $r_{min,max}$ and d_{max}
Output: Placement locations (x_i, y_i) for G
1: Initialize random population
2: Overlap removal
3: Register collapsing
4: Evaluate the population
while *Termination criterion is not satisfied* **do**
  5: Select chromosomes by reproduction procedure
  6: Perform Order Multiple crossover technique with probability P_c
  7: Mutate
  8: Overlap removal
  9: Register collapsing
  10: Evaluate the population
end

Algorithm 6: Proposed genetic placement algorithm for generating highly optimized structured latch clusters.

new population is evaluated. Termination of the GA is based on iteration count with details provided in Section 5.8.

5.5 Structured Template Generation

The placement search from the prior section exhaustively generated a set T of structured placement solutions for all input combinations including latch and LCB types, sizes, densities, ratios, skew and delay constraints, and latch cluster sizes. As Section 5.2.2 shows, clearly it is not practical to store all placement solutions for all input combinations and in fact, many solutions are redundant.

Figure 5.6 illustrates such an example. As Figure 5.6 shows, for all

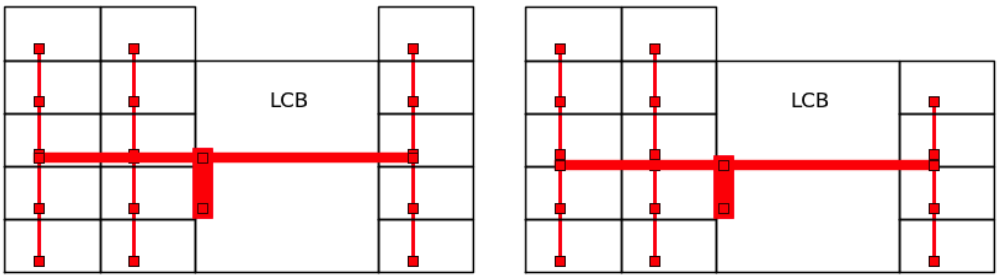


Figure 5.6: Two structured latch cluster templates are shown with clock routing in red where (a), is a 15 latch template example and (b) is a 14 latch template example. Template (b) is a redundant template because (a) has more latches and the equivalent latches between (a) and (b) have identical placement.

LCB and latch placement locations in (b), (a) has identical placements. By storing the difference between (a) and (b), template (b) no longer adds any additional placement information compared to the template in (a). Therefore (b) is redundant[1]. This section presents a method to significantly reduce the cardinality of T by removing redundancy.

5.5.1 Set-Theoretic Template Annotation

Given a structured latch cluster $G = (L, LCB)$ with latches L and LCB, each with placement locations (x_i, y_i) $0 \leq i \leq |L|$, Definition 9 formalizes the definition of redundancy.

Definition 9. Given templates $(M, N) \in T$, M is redundant $\iff$

[1]The proposed approach is a form of delta-encoding that greatly reduces data redundancy by storing information in the form of differences without loss of information.

- $|M| < |N|$,
- $\forall(x_i, y_i)$ LCB and latch placement locations in M, there $\exists$ identical (x_i, y_i) locations in N
- the bounding box width of $M = N$
- the bounding by height of $M = N$.

In this work, $M \lhd N$ denotes template M is made redundant by template N. By simply annotating template N with the set-theoretic difference $\xi = N \diagdown M$, it is possible to generate the placement solution for both N and M with a single template representation. Qualitatively, ξ is the set of latches in N that are not in M. Definition 10 formally defines ξ for two templates M and N.

Definition 10. The set-theoretic difference ξ for templates $(M, N) \in T$, where $M \lhd N$ is $N \diagdown M = \{l \in N | l \notin M\}$ $\forall l \in L_{M \cup N}$

Proving that generating both placement solutions is possible using a single template that contains ξ is trivial.

Proof. For templates $(M, N) \in T$ s.t. $M \lhd N$ and the relative compliment of N in M is ξ, let set $C = N \diagdown \xi$. The set C now contains all latches in N except those from the set ξ. But by definition 10, all latches $l \notin \xi$ is the set M. □

By storing ξ for each redundant template in this manner, it is possible to fully reconstruct all optimized structured latch cluster placement solutions

from a single annotated template. The overall reduction algorithm is presented next.

5.5.2 Proposed Redundancy Removal Algorithm

Algorithm 7 presents the proposed method for redundancy removal from T without loss of quality. Starting with an initial template set T, step (1) groups all templates into bins containing equal width and height. Then, for each group, step (2) sorts those templates by latch count. There can arise cases where multiple solutions exist with equivalent register count and total capacitance. In that case, step (3), an arbitrary solution is chosen and the others discarded. Step (4) assigns the current template being evaluated to the first element in w. The algorithm then checks for redundancy between the current template $curr_t$ and the next element in w. If template w_i is redundant, step (5) calculates the set-theoretic difference $t[i]_\xi$ and step (6) annotates $curr_t$ with $t[i]_\xi$. Finally, step (7) drops the redundant template $t[i]$ and the algorithm continues to the next template in w. If template $t[i]$ is not redundant with $curr_t$, annotation of $curr_t$ is complete. Thus, $curr_t$ is assigned the new template $t[i]$ and the algorithm moves to the next template in w. The proposed approach offers three key benefits. First, Algorithm 7 is a form of delta encoding offering no data loss. Second, it is very fast, running in $(O(n \cdot log(n)))$. Third, it is parallelizable on template sets W. As results will show, the proposed approach significantly reduces the cardinality of T.

Function: $TemplateReduce(E, F)$;
Input: Initial Template Set T
Output: Reduced Template Set T annotated with ξ
1. Build template sets W of groups with equal width and height;
for *each* $w \in W$ **do**
    2. $\forall$ templates $t \in w$, sort in descending order of latch count;
    3. Remove duplicates from w;
    4. Assign $curr_t = w_0$;
    for $\{i = 1,\ i < |w|,\ i++\}$ **do**
        if $t[i] \lhd curr_t$ **then**
            5. Calculate $t[i]_\xi$;
            6. Annoate $curr_t$ with $t[i]_\xi$;
            7. Drop $t[i]$ from T;
        else
            8. $curr_t = t[i]$;
        end
    end
end
Algorithm 7: Algorithm for redundancy removal from T.

5.6 Decision Tree Induction for Structured Template Selection

Storing a comprehensive library mapping all inputs to a particular optimized template and ξ is neither practical nor required. Additionally, it is not possible to anticipate every situation for every latch cluster considering there are hundreds of thousands in a full design. The physical design (PD) flow needs a "decision" algorithm to quickly choose the best template given an unknown set of input requirements. Machine learning techniques offer many effective approaches including: neural networks (ANN) [85], support vector machines (SVM) [86] [22], and decision trees [45]. Additionally, many have

been used to successfully solve other design automation challenges [88][21].

Neural networks and SVMs are popular techniques because both model nonlinear relationships between the input variables and handle inter-variable interactions. However, both offer a number of drawbacks. First, both ANN and SVM cannot natively handle categorical variables with multiple classes. Arbitrary value thresholds are required for categorical discrimination (such as latch and LCB types) but decision trees handle this elegantly. Second, ANN and SVM do not present comprehensible models in a way designers will understand. Lastly, it can be difficult to incorporate ANN or SVM models into existing code without a dedicated interpreter requiring further library development. Decision trees however naturally convert to if...then...else statements leading to easy implementation or as a reference for designers. As such, this work proposes machine learning based decision tree induction for structured template selection.

5.6.1 Decision Tree Classification Overview

A decision tree classifier is a method that predicts a target class F, in this case a particular template, based on an input vector E where $(E, F) = (e_1, e_2, e_3, ..., e_k, F)$ with $(1 \leq k \leq |E|)$. For this work, E is the input parameters defined in Sec. 5.4. A decision tree learns by recursively partitioning the source data into subsequent subsets based on the attribute test. For this application, a decision tree classifier is particularly effective because:

1. Classifying with decision trees is a nonparametric approach meaning no

prior probability distributions are required.

2. Finding an optimal decision tree is NP-complete [49] but in practice the greedy induction approaches are very effective.

3. Decisioning after training is very fast with worst case $O(\omega)$ [49], where ω is the depth of the tree.

The base decision tree induction algorithm is presented in Algorithm 8. It is similar to prior techniques [45] with a critical enhancement for the splitting index. Step 1 creates a new node with either a test condition or a class label (in this case a specific template) and Step 2 and Step 3 classify and return the final decision in the case where the stopping criterion is met. In this work, a minimum number of test records within a leaf defines the stopping criterion. Let V be the set of possible templates (class labels) in node $leaf$, the leaf is labeled with the class that has the majority number of training records. Thresholding in this manner avoids over-fitting the data (avoids generalization errors). If the minimum criterion is not yet met, a new node (Step 4) called *root* is created and Step 5 finds the best split based on the novel measure presented in Section 5.6.2. Then, for each possible outcome (class label) in *root*, Steps 7, 8, and 9 setup the recursive call for evaluating a new node.

5.6.2 Novel Placement Similarity Impurity Measure

The key enhancement to decision tree induction for structured latch templates is the observation that, inter class error rates are not consistent.

Function $TreeBuild(E, F)$;
Input: Training records (E)
Output: Decision Tree Model
if $stopping\_condition(E, F) = TRUE$ **then**
    1. create a new node from the $leaf$;
    2. classify the $leaf$;
    3. return $leaf$;
else
    4. create a node $root$;
    5. find the best split and set it equal to the $root$ test condition;
    6. let V = the set of possible outcome test conditions of the $root$ node;
    for *each* $v \in V$ **do**
        7. E_v = training records given the $root$ test condition;
        8. $child = TreeBuild(E_v, F)$;
        9. add child as descendent of root and label;
    end
end
10. return $root$

Algorithm 8: Decision tree induction algorithm.

In other words, if two templates are similar, the capacitance loss in choosing the wrong one is lower than choosing a template that is less similar from the correct solution. The magnitude of the impurity metric should be greater in the case where two class solutions are very different and smaller in the case where the two class solutions are very similar. Therefore, a distance measure is proposed that defines an impurity measure in terms of the similarity in placement solutions between two templates. Figure 5.6 will be used as an example and assume both are in the same class. Using the ordered candidate representation, the bottom left latch in both (a) and (b) will be labeled as the first latch. The second will be the next latch directly to the right of it. This

continues for all latches in that row and then moves to the next row.

Once all latches are labeled in this order, it is possible to compare the latch placement at a particular location between two templates. For the example in Figure 5.6, both latches have the same position for the first latch so the class average position μ would equal the position of that latch with a zero variance σ. Additionally, since template (a) has more latches than template (b), the μ for the position of the missing latch in (b) is simply the latch (a) position. This work splits each node into two groups. The average and variance for each template set for the two groups is calculated to measure the impurity. Let μ_n^i, $1 \leq n \leq |L|$ be the mean Euclidian distance belonging to class i at a given latch n at a given node. Similarly, σ_n^i is the variance at that position. Then, for each input variable E $1 \leq k \leq |E|$, Equation (5.6) calculates the information gain achieved by splitting the node based on input k. By multiplying the inverse variance, classes with large variations have reduced likelihood of being selected.

$$\Delta^{i,j} = \sum_{k=0}^{K} \left(\sqrt{\sum_{n=0}^{N} (\mu_n^i - \mu_n^j)^2 * (1 + \sigma_n^i + \sigma_n^j)^{-1}} \right) \tag{5.6}$$

5.6.3 Supervised Learning Model Build Flow

To classify and evaluate the compact and run-time efficient template selection algorithm, the flow in Figure 5.7 is proposed. In Step A, the data learning algorithms are applied over a relatively small set of input patterns with known templates. Since they are built *a priori* at a one time cost, the

CPU run-time penalty is negligible.

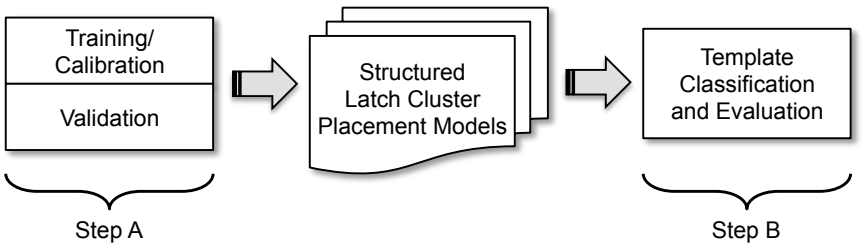


Figure 5.7: Major steps to build and apply the decision-tree learning model

There are 3 major procedures involved in this step: (1) training is the process where the learning algorithms optimize structured template accuracies; (2) calibration process further improves the accuracy by adjusting the pruning rate, (3) validation process is performed over a relatively large set of known design patterns exclusive from (1) to assure the balance of learning accuracies between training data and unknown testing data, especially in Step B. Once Step A is completed, in Step B the data learning model is applied directly to classify and evaluate new unknown input patterns. All of this is done at a one time cost per technology library. To quantify the learning performance, the following accuracies are defined:

Definition 11. Template evaluation accuracy: the rate of correctly selected templates over the total number of predicted inputs.

In the next section, an overview of how the template solutions integrate with a modern physical design flow is presented.

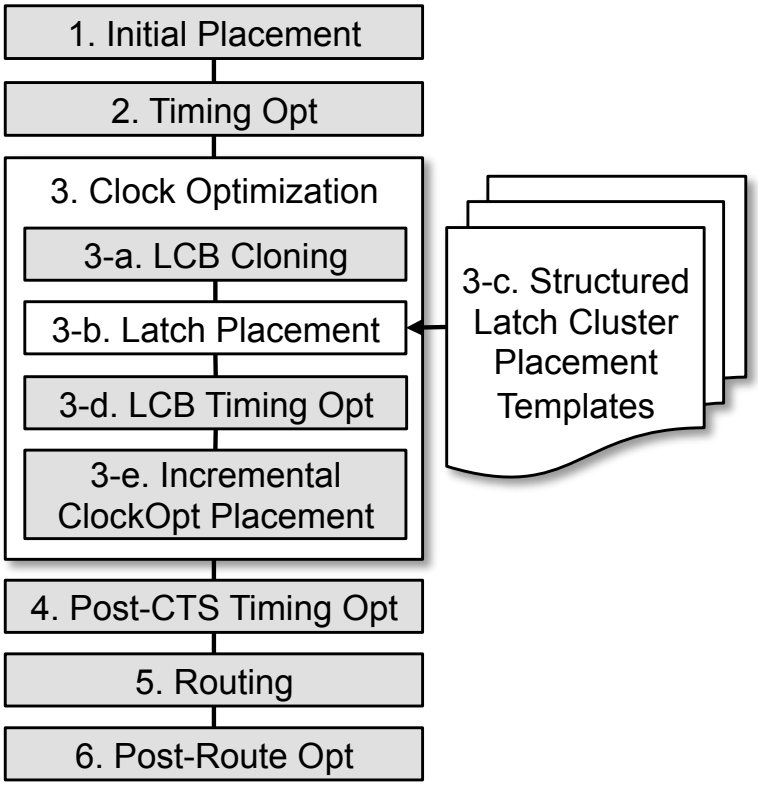


Figure 5.8: Modern physical design flow illustrating each stage of the automation process. The shaded boxes are the current flow. Step 3-b displays where the optimized placement templates integrate into the flow instead of the conventional clustered approach.

5.7 Overall Physical Design Flow

At this point, a set of optimized placement templates have been identified and a machine learning decision model has been trained to correctly select a good template given an arbitrary input during the design flow. This is done at a one time cost per standard cell library. Before presenting the impact of the proposed techneques, it is worthwhile to provide a brief overview of how the

model integrates into a modern state-of-the-art physical design flow. Figure 5.8 shows steps 1-6 in such a flow [59][60]. This chapter integrates into clock optimization (Step 3) which consists of four major components: LCB cloning, latch placement, LCB timing optimization, and incremental clock optimization placement. The LCB cloning, (3-a) inserts redundant LCBs to limit the fanout and assigns latches to a single LCB. This step is important because maximum slew constraints limit the number of latches per LCB. The next stage, latch placement (3-b), modifies the position of the latches by reducing the distance between each latch and the assigned LCB. It is at this stage the latch cluster placement will be defined by the proposed structured templates. Finishing up clock optimization, steps (3-c,d) attempt to correct the timing disruption caused by latch movements within the design. The following section demonstrates the effectiveness of the proposed methodology by evaluating the capacitance ratio of the structured latch clusters verses conventional approaches. As such, Def. 12 defines the capacitance ratio used to compare the different approaches in the following section.

Definition 12. Capacitance Ratio is defined as the routed capacitance of a structured template divided by the routed capacitance of an equivalent clustered latch cluster.

5.8 Experimental Results

The proposed framework was implemented in C++ and the placement search, Section 5.4, was successively called over all input combinations. Input parameters, latch types and LCB sizes were selected based on library requirements from a state-of-the-art 22nm technology library. All resistance and capacitance values were taken from this library and clock routing and delay measurements remained consistent across all experiments. To quantify the effectiveness of the proposed methodology, results are compared against the latch clustering technique as presented in [62] and register banking in [28]. The experimental results are presented in three stages. First, a overview of the template generation inputs are outlined and the effectiveness of the template reduction from Section 5.5 is presented. Then, the capacitance reduction from using the generated templates is presented. Finally, the results are presented for the decision tree induction model demonstrating the effectiveness of the approach to identify good placement templates on unknown input combinations.

5.8.1 Template Generation and Redundancy Removal Results

The proposed methodology developed a set of latch placement templates optimized for minimizing local clock tree capacitance. These templates were generated over a large operating range of values expected to be encountered during a design lifecycle. Input parameters to the placement search were selected, derived from a multi-Ghz design, as follows. Latch clusters of size 7

to 67 were evaluated with four different LCB sizes. Five different bounding box ratios [0.1 : 0.5), [0.5 : 1.0), [0.5 : 1.5), [1.0 : 1.5) and [1.5 : 10) and five different density ranges [0.7 : 1.0], [0.75 : 1.0], [0.8 : 1.0], [0.85 : 1.0] and [0.9 : 1.0] were evaluated against four skew $4(ps)$, $5(ps)$, $6(ps)$ and $7(ps)$ and four max delay requirements $5(ps)$, $6(ps)$, $7(ps)$ and $8(ps)$. Placement solutions were generated exhaustively for *all input combinations resulting in* $96,000$ *initial solutions*. For each, the GA termination criterion was bounded at 1 million populations with 100 candidates per population and a constant mutation rate of 1%. The λ value from Equation (5.5) was held constant at 0.05.

After placement solutions were generated, redundant placement cluster removal was applied as presented in Section 5.5. The initial cardinality of the template set T was $96,000$. By removing invalid solutions and applying redundant template removal, the total number of templates reduced to only 534 total annotated templates. This *resulted in only 0.56% of the original possibilities* clearly demonstrating the effectiveness of the proposed approach.

5.8.2 Advantage of the Proposed Structured Latch Placement Templates

Because capacitance reduction directly reduces chip power, after generating the template set, four latch placement approaches are compared in terms of total routed local clock capacitance. The four techniques are summarized as follows:

- *Clustered*: Local clock tree (LCT) capacitances are different for each instance of a clustered solution therefore the average routed LCT of one hundred latch clusters is used as a baseline.
- *Banking*: A bank of latches is generated for all latch counts within that range of parameter requirements.
- *Structured*: This is the proposed approach with optimized structured latch templates selected based on the input parameter set.
- *Trained*: This is the template selected based on decision tree induction.

All approaches are compared relative to the conventional clustered approach using the capacitance ratio. Clearly presenting every input combination is not feasible. As such, Tables 5.1 and 5.2 present four sets of ratio sizes across selected latch counts with the last row displaying the average for each of the techniques.

Table 5.1: Total capacitance ratio comparing clustered, banked, structured, and trained model techniques with density ratio greater than 0.7. The banked and structured columns are the results when always selecting the correct template. The trained column is the results produced from the learning model.

	$0.5 < \beta < 1.0$				$0.1 < \beta < 0.5$			
Latch #	Clustered	Banked	Structured	Trained	Clustered	Banked	Structured	Trained
7	1.00	1.99	0.85	0.85	1.00	0.99	0.87	0.90
11	1.00	0.98	0.81	0.83	1.00	0.94	0.83	0.83
15	1.00	0.97	0.83	0.84	1.00	0.96	0.84	0.86
19	1.00	0.91	0.80	0.80	1.00	0.93	0.81	0.81
23	1.00	0.91	0.79	0.79	1.00	0.98	0.79	0.81
27	1.00	0.90	0.78	0.78	1.00	0.96	0.78	0.78
31	1.00	0.83	0.75	0.83	1.00	0.88	0.77	0.77
35	1.00	0.87	0.72	0.87	1.00	0.88	0.74	0.74
39	1.00	0.87	0.68	0.87	1.00	0.87	0.73	0.73
43	1.00	0.84	0.68	0.85	1.00	0.83	0.73	0.75
47	1.00	0.83	0.65	0.65	1.00	0.82	0.71	0.71
51	1.00	0.80	0.63	0.63	1.00	0.79	0.70	0.70
55	1.00	0.80	0.61	0.61	1.00	0.79	0.69	0.69
59	1.00	0.77	0.60	0.63	1.00	0.77	0.68	0.68
63	1.00	0.78	0.59	0.64	1.00	0.77	0.67	0.69
67	1.00	0.78	0.58	0.61	1.00	0.75	0.67	0.67
Ave.	1.00	0.93	0.71	0.76	1.00	0.87	0.75	0.76

From the selected samples, the structured approach uses 0.71 of the clock tree capacitance compared to a clustered solution with a beta ratio between 1.5 and 10. Assuming clock power consumes up to 50% of a design and the local clock tree consists of 30% of the total clock power, this results in *roughly a 4% reduction in total power* when compared to the clustered approach. When compared to the banked approach, of the same shape, it results

Table 5.2: Total capacitance ratio comparing clustered, banked, structured, and trained model techniques with density ratio greater than 0.7. The banked and structured columns are the results when always selecting the correct template. The trained column is the results produced from the learning model.

	$0.5 < \beta < 1.0$				$0.1 < \beta < 0.5$			
Latch #	Clustered	Banked	Structured	Trained	Clustered	Banked	Structured	Trained
7	1.00	0.99	0.89	0.89	1.00	0.96	0.86	0.86
11	1.00	0.98	0.85	0.85	1.00	0.98	0.88	0.88
15	1.00	0.97	0.84	0.88	1.00	0.99	0.88	0.90
19	1.00	0.96	0.85	0.85	1.00	0.97	0.90	0.90
23	1.00	0.96	0.85	0.85	1.00	0.95	0.89	0.89
27	1.00	0.93	0.82	0.83	1.00	0.96	0.88	0.88
31	1.00	0.88	0.81	0.81	1.00	0.88	0.82	0.82
35	1.00	0.88	0.75	0.75	1.00	0.87	0.82	0.82
39	1.00	0.87	0.75	0.75	1.00	0.87	0.81	0.81
43	1.00	0.84	0.74	0.74	1.00	0.83	0.77	0.80
47	1.00	0.83	0.71	0.75	1.00	0.82	0.76	0.76
51	1.00	0.80	0.69	0.69	1.00	0.79	0.73	0.73
55	1.00	0.81	0.68	0.68	1.00	0.79	0.70	0.70
59	1.00	0.78	0.67	0.67	1.00	0.77	0.71	0.73
63	1.00	0.78	0.64	0.67	1.00	0.76	0.70	0.70
67	1.00	0.78	0.64	0.64	1.00	0.75	0.69	0.69
Ave.	1.00	0.88	0.76	0.77	1.00	0.87	0.80	0.80

in roughly over a 3% reduction in power. Assuming a 100-watt microprocessor, that equates to a potential 2 to 4 watt savings. At a beta ratio between 0.1 and .5, structured used 0.80 of the clustered solution resulting in up to 3% reduction in dynamic power and still produced roughly 1 watt of savings for a 100-watt microprocessor. Though templates will be technology and possibly chip specific, results clearly show the potential power savings.

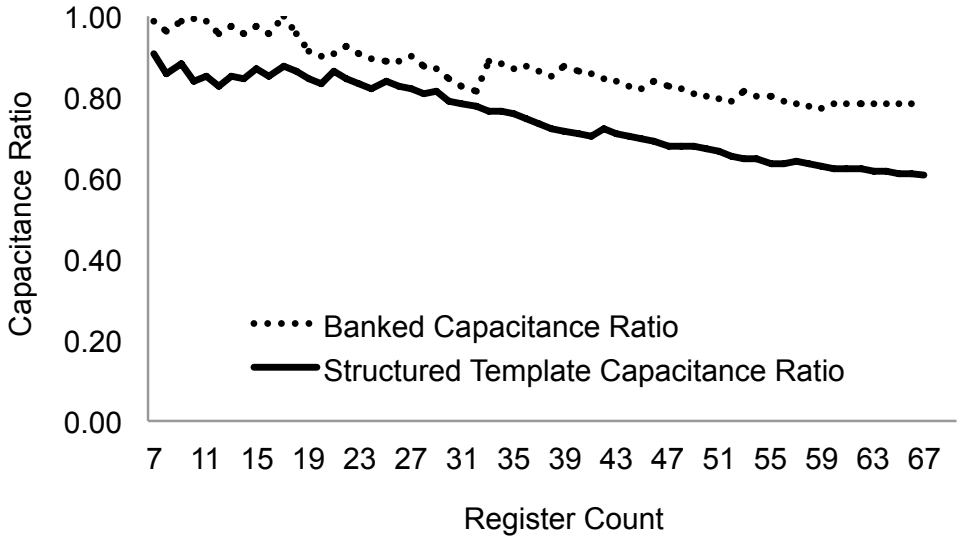


Figure 5.9: Capacitance ratio of banking versus the structured template methodologies corresponding to columns 2 and 3 in Table 5.2 .

To further illustrate the range of capacitance savings, Figure 5.9 plots the capacitance ratio of both banking versus structured template methodologies corresponding to columns 2 and 3 in Table 5.2. For local register group between 19 to 31, the two methodologies produce very similar routed capacitance. However, for latch groups of larger sizes, there is significant benefit to applying the structured latch template methodology. By increasing the number of available templates, capacitance reduces for a wider range of inputs. Next, results are presented showing it is possible to train a decision tree to effectively select the appropriate template solution.

5.8.3 Decision Tree Induction Results

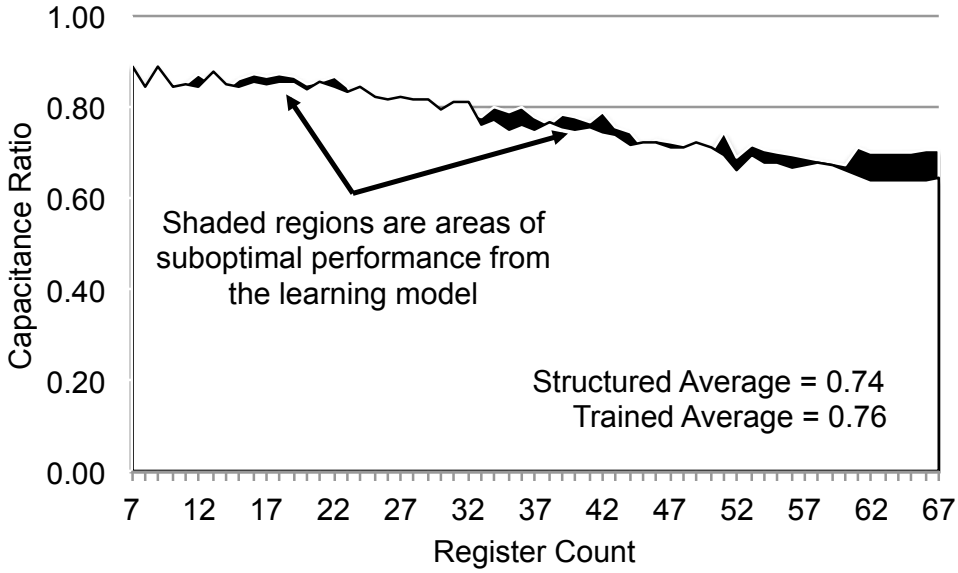


Figure 5.10: Capacitance ratio of structured verses trained solutions. The shaded regions illustrate the latch cluster sizes where the learning model on average did not select the correct template.

This subsection presents the results from the proposed method in Section 5.6 where a decision tree was described with a modified distance function. As the results will show, it is not necessary to record all possible input combinations and corresponding templates in a huge lookup table. Instead, it is possible to apply decision tree induction that is implemented as a series of if..and..else statements. As Figure 5.7 shows, the model building process happens in two parts. First, on a small sample of data, training and validation is applied using the proposed training methodology. In this work, the training set is randomly sampled from 75% of the total data (75% training and

25% validation). Care must be made to ensure representative templates of all class labels are represented. Therefore, if there are not enough samples from a particular class, samples from an overpopulated class are discarded and a new sample is selected at random. Once training is complete, the evaluation step (Figure 5.7 Step B) runs the model on untrained data samples (the remaining 25% of the population).

Using the proposed approach, *the classification accuracy was 0.918*, indicating the correct template was selected approximately 91% of the time across all validation samples. To further illustrate the effectiveness of this approach, Figure 5.10 displays a graph of the average capacitance ratio of the correct template and the template selected by the learning model. The latch cluster size is presented on the x-axis. The y-axis is the average capacitance ratio of all templates from the untrained data samples. The shaded regions indicate where the learning model selects a template other than one identified as the "optimal" template.

5.8.4 Runtime Results

The proposed flow is precharacterizing optimized placement templates at a one time cost for each technology library thus only template selection influences actual PD runtimes. For brevity, results are presented for all sections. Initial placement search, Section 5.4 averaged 562 seconds(s) per template but is fully parallelizable with total iterations bounded to 1 million. Infeasible template solutions were discarded and not included in the average search

time. Template redundancy removal, Section 5.5, executed in 108(s) and model training, Section 5.6, completed in 419(s).

The key runtime impact is the amount of time to evaluate a decision tree. Decisioning, based on an unknown input during PD, ranges between *2 and 7 milliseconds (ms)* and averages 5.3(ms) depending on the number of branches between the root node and the leaf. Additionally, template selection is fully parallelizable per latch cluster. Thus, the proposed approach offers an extremely fast and effective method for latch cluster placement because the optimization of the template is one time for all designs.

5.9 Summary

Microprocessor power design constraints significantly limit performance enhancements. By minimizing routed capacitance of the latch clusters, significant power savings is possible on the local clock tree which can contribute up to 30% of the total power in a design. First, a set of structured templates is generated. Then, using machine learning based decision tree induction, a learning model is trained using a novel distance measurement. The decision tree enables the design flow to "decide", based on arbitrary input combinations, the best template to use during clock optimization of the physical design flow. Using these approaches, this chapter shows it is possible to reduce local clock tree capacitance by 20-30% compared to the state-of-the-art. Assuming the clock power contributes 50% of the total dynamic power, on a 100-watt microprocessor that results in roughly between a 1 and 4 watt reduction.

Chapter 6

Conclusions

This section summarizes the impact of the proposed work and discusses challenging future research opportunities. The key theme throughout this work is carefully identifying suboptimality during placement then developing methods to integrate structure into the placement flow to produce significant quality improvements. As this work has shown, high fanout net suboptimality impacts many aspects of physical design including datapath automation and clock power optimization.

6.1 Summary

Structured datapath circuits were the first to be explored. Many attempts have been made in the last 40 years to close the quality gap between manual and automated placement of datapaths and other regular structures. Nevertheless, a common assumption still prevails among IC designers that circuits with high regularity require manual placement.

This work showed that the primary optimization objective of modern state-of-the-art placement algorithms, HPWL, can mislead placers on datapath-oriented designs. In particular, compressing placement of high-fanout

nets to lower the overall HPWL disrupts the regularity of placement and undermines its Steiner wirelength, which is known to better correlate with routed wirelength. Based on this observation, a unified framework was developed to enhance current random-logic placers to better handle designs containing datapath logic seamlessly integrating alignment constraints into a state-of-the-art placement engine. Experimental results show at least a 28% improvement in total StWL compared with the state-of-the-art academic placers for the ISPD 2011 Datapath Benchmark Suite and a 5.8% average improvement in total StWL for industrial hybrid designs. Additionally, significant improvement in routability is achievable through datapath alignment.

This work also showed that power reduction is possible through structured placement techniques. By minimizing routed capacitance of the latch clusters, this work shows that significant power savings is possible on the local clock tree which can contribute up to 30% of the total power in a design. First, a set of structured templates is generated. Then, using machine learning based decision tree induction, a learning model is trained using a novel distance measurement. The decision tree enables the design flow to decide, based on arbitrary input combinations, the best template to use during clock optimization of the physical design flow. Using these approaches, this work shows it is possible to reduce local clock tree capacitance by 20-30% compared to the state-of-the-art. Assuming the clock power contributes 50% of the total dynamic power, on a 100-watt microprocessor that results in roughly between a 1 and 4 watt reduction.

6.2 Open Challenges

This work shows it is possible to bridge the gap between custom and automated design flows but it is just the start. Though results are impressive, there is significant opportunity for further placement improvements. Exasperating this complex issue is the continuing trend for larger and larger designs blending traditional ASIC style logic with some embedded datapath structures. This hybrid design style [90, 88] adds significant complexity to the datapath design problem because of the need to both identify (classify) the structures before optimizing the place and route.

To truly automate structured placement solutions, research needs to understand more thoroughly why manually design solutions work and how to automatically detect those cases. With the future being hybrid designs, research needs to address

- at what stage datapath classification should occur,
- how to effectively classify different datapath design structures without runtime and global design constraint impact,
- effective optimization techniques for each of the design styles after classification.

From practical experience, three common design styles have been encountered that are commonly lumped under the umbrella of "datapath" and still require manual design intervention.

1. **Logical Regularity** The first major classification are circuits that require logic level optimization to effectively meet timing. The most commonly referenced circuit of this type are adders but include multipliers, dividers, and other complex arithmetic operations. This class of circuits requires identification and optimization at the logic level to effectively meet design constraints.

2. **Physical Regularity** The second class of datapath logic, similar to the benchmarks presented in [92] are highly regular structures such as rotators or MUXes that improve significantly through physical alignment. As shown in [92], general academic placers are terrible at quickly solving even the most basic datapath problem instance of this style. One reason is that optimizing half-perimeter bounding box wirelength is the wrong metric for datapath nets and for the enable nets. Some resent work such as [14, 90] has shown promising improvements in this logic style though much research still needs to be done. Additionally, approaches such as [88] seem to show that it is possible to incrementally modify placement of these structures during placement to improve either wirelength or congestion.

3. **Hierarchical Regularity** The third class of structures often referred to as datapath logic are semi-structured designs with hierarchical regularity including queues, standard cell content addressable memories (CAMs) or large data buses going through regular pipeline stages. In all of these

circuits, there can exist other datapath style logics but in general they are not strictly bit-stack regular such as those in [92]. In a modern microprocessor, a store queue is an example of this structure. A store queue design queues the list of outstanding store instructions where a single entry within the queue contains the state information and status for one microprocessor store instruction. Within the queue there can be many smaller datapath style circuits such as comparators or adders, but careful clustering and placement of the hierarchical structure is critical for overall design quality.

Two key questions must be addressed for each style before significant industrial adoption.

1. How does one quickly find these structures automatically (search and classify these structures) including what stage this should be done (ex: before synthesis, during placement...)

2. Once found, what is the right optimization approach balancing global and local constraints (the optimization problem)

For each of these design styles, significant future research is needed to effectively automate place and route. Additionally, in the past automated solutions try and mimic manual placement solutions for these design styles.

This work demonstrates the great potential of a datapath aware placement framework with the overriding goal to develop a fully automated extraction and placement flow. Additionally it shows that significant power savings

is possible through the use of structured placement for local clock trees. However, this is just the beginning. Significant research opportunity still exists in which another decade may not be enough time to truly solve these vexing problems.

Bibliography

[1] S. N. Adya and I. L. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 12–17, 2002.

[2] C. J. Alpert, Zhuo Li, Gi-Joon Nam, C. N. Sze, N. Viswanathan, and S. I. Ward. Placement: Hot or not? In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 283–290, 2012.

[3] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar (eds.), editors. *Handbook of Algorithms for VLSI Physical Design Automation.* Springer, New York, NY, 2008.

[4] C.J. Alpert, S.K. Karandikar, Zhuo Li, Gi-Joon Nam, S.T. Quay, Haoxing Ren, C.N. Sze, P.G. Villarrubia, and M.C. Yildiz. Techniques for fast physical synthesis. *Proceedings of the IEEE*, 95(3):573–599, 2007.

[5] J. E. Beasley. *Advances in Linear and Integer Programming.* Oxford University Press, Inc., New York, NY, 1996.

[6] Tony F. Chan, Jason Cong, Joseph R Shinnerl, Kenton Sze, and Min Xie. mPL6: enhanced multilevel mixed-size placement. In *Proceedings*

ACM International Symposium on Physical Design (ISPD), pages 212–214, 2006.

[7] C.-C. Chang, J. Cong, D. Z. Pan, and X. Yuan. Multilevel global placement with congestion control. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):395–409, 2003.

[8] Chin-Chih Chang, Jason Cong, Michail Romesis, and Min Xie. Optimality and scalability study of existing placement algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(4):537–549, 2004.

[9] Ting-Hai Chao, Yu-Chin Hsu, Jan-Ming Ho, Kenneth D. Boese, and Andrew B. Kahng. Zero skew clock routing with minimum wirelength. *IEEE Transactions on Circuits and Systems II*, 39(11):799–814, 1992.

[10] Rishi Chaturvedi and Jiang Hu. Buffered clock tree for high quality ic design. In *International Symposium on Quality Electronic Design (ISQED)*, pages 381–386, 2004.

[11] Tung-Chieh Chen, Minsik Cho, David Z. Pan, and Yao-Wen Chang. Metal-density driven placement for cmp variation and routability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(12):2145–2155, 2008.

[12] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. A high-quality mixed-size analytical placer consid-

ering preplaced blocks and density constraints. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 187–192, 2006.

[13] M. Cho and D. Z. Pan. Boxrouter: A new global router based on box expansion. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2130–2143, 2007.

[14] Sheng Chou, Meng-Kai Hsu, and Yao-Wen Chang. Structure-aware placement for datapath-intensive circuit designs. In *ACM/IEEE Design Automation Conference (DAC)*, pages 762–767. ACM, 2012.

[15] A. Chowdhary, S. Kale, P.K. Saripella, N.K. Sehgal, and R.K. Gupta. Extraction of functional regularity in datapath circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1279–1296, 1999.

[16] Vincent A. Cicirello. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In *Proceedings Genetic and Evolutionary Computation Conference*, pages 1125–1131, 2006.

[17] S. Das and S.P. Khatri. A merged synthesis technique for fast arithmetic blocks involving sum-of-products and shifters. In *VLSI Design, 21st International Conference on*, pages 572–579, 2008.

[18] S. Das and S.P. Khatri. A timing-driven approach to synthesize fast barrel shifters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 55(1):31–35, 2008.

[19] R. L. Davis. Uniform shift networks. *Computer*, 7(9):60–71, 1974.

[20] D.J. Deleganes, M. Barany, G. Geannopoulos, K. Kreitzer, M. Morrise, D. Milliron, A.P. Singh, and S. Wijeratne. Low-voltage swing logic circuits for a pentium 4 processor integer core. *IEEE Journal of Solid-State Circuits*, 40(1):36–43, 2005.

[21] Duo Ding, Bei Yu, Joydeep Ghosh, and David Z. Pan. Epic: Efficient prediction of ic manufacturing hotspots with a unified meta-classification formulation. In *Asian and South Pacific Design Automation Conference (ASPDAC)*, pages 263–270, 2012.

[22] Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working Set Selection Using Second Order Information for Training Support Vector Machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

[23] S. Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Transactions on Circuits and Systems*, 28(1):12–18, 1981.

[24] Lars W. Hagen, Dennis J.-H. Huang, and Andrew B. Kahng. Quantified suboptimality of VLSI layout heuristics. In *ACM/IEEE Design*

Automation Conference (DAC), pages 216–221, 1995.

[25] J.L. Hennessy and D. A. Patterson, editors. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1996.

[26] M. A. Hillebrand, T. Schurger, and P. M. Seidel. How to half wire lengths in the layout of cyclic shifters. In *Fourteenth International Conference on VLSI Design*, pages 339–344, 2001.

[27] Pei-Hsin Ho. Industrial clock design. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 139–140, 2009.

[28] Wenting Hou, Dick Liu, and Pei-Hsin Ho. Automatic register banking for low-power clock trees. In *International Symposium on Quality Electronic Design (ISQED)*, pages 647–652, 2009.

[29] J. Hu, J. A. Roy, and I. L. Markov. Completing high-quality global routes. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 35–41, 2010.

[30] Liang Huang, Yici Cai, Qiang Zhou, Xianlong Hong, Jiang Hu, and Yongqiang Lu. Clock network minimization methodology based on incremental placement. In *Asian and South Pacific Design Automation Conference (ASPDAC)*, pages 99–102, 2005.

[31] Paolo Ienne and Alexander GrieBing. Practical experiences with standard-cell based datapath design tools. In *ACM/IEEE Design Automation Conference (DAC)*, pages 396–401, 1998.

[32] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Workshop on Analytic Algorithms and Combinatorics*, pages 135–149, 2007.

[33] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 891–898, 2005.

[34] Andrew B Kahng and Qinke Wang. Implementation and extensibility of an analytic placer. *Proceedings ACM International Symposium on Physical Design (ISPD)*, 24(5):18–25, 2004.

[35] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):69–79, 1999.

[36] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(1):291–307, 1970.

[37] M.-C. Kim, D.-J. Lee, and I. L. Markov. SimPL: an effective placement algorithm. *IEEE Transactions on Computer-Aided Design of Integrated*

Circuits and Systems, 31(1):50–60, 2012.

[38] M.-C. Kim and I. L. Markov. ComPLx: a competitive primal-dual lagrange optimization for global placement. In *ACM/IEEE Design Automation Conference (DAC)*, pages 747–752, 2012.

[39] Israel Koren, editor. *Computer Arithmetic Algorithms*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1993.

[40] Thomas Kutzschebauch and Leon Stok. Regularity driven logic synthesis. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 439–446, 2000.

[41] l. Kozhaya, P. Restle, and Haifeng Qian. Myth busters: Microprocessor clocking is from mars, asics clocking is from venus. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 271–275, 2011.

[42] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479, 1971.

[43] Dong-Jin Lee, Myung-Chul Kim, and Igor L. Markov. Low-power clock trees for cpus. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 444–451, 2010.

[44] Q. Liu and M. Marek-Sadowska. Pre-layout physical connectivity predictions with applications in clustering, placement and logic synthe-

sis. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 31–37, 2005.

[45] Susan Lomax and Sunil Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys*, 45(2), 2013.

[46] A. Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981.

[47] Luks and Eugene M. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings IEEE Symposium on Foundations of Comp. Science*, pages 42–49, 1980.

[48] T. Luo, H. Ren, C.J. Alpert, and D.Z. Pan. Computational geometry based placement migration. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 41–47, 2005.

[49] Mikhail Ju. Moshkov. Time complexity of decision trees. In *Transactions on Rough Sets III*, volume 3400 of *Lecture Notes in Computer Science*, pages 244–459. Springer Berlin Heidelberg, 2005.

[50] Silvia M. Muller and Wolfgang Paul. *Computer Architecture: Complexity and Correctness*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[51] G.-J. Nam. Ispd 2006 placement contest: Benchmark suite and results. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, page 167, 2006.

[52] G.-J. Nam, C. J. Alpert, P. G. Villarrubia, B. B. Winter, and M. C. Yildiz. The ispd 2005 placement contest and benchmark suite. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 216–220, 2005.

[53] Gi-Joon Nam and Jason Cong, editors. *Modern Circuit Placement: Best Practices and Results*. Springer, New York, NY, 2007.

[54] Raymond X. T. Nijssen and Jochen A. G. Jess. Two-dimensional datapath regularity extraction. In *ACM/IFIP Workshop on Logic and Architecture Synthesis*, pages 110–117, 1996.

[55] Satoshi Ono and Patrick H. Madden. On structure and suboptimality in placement. In *Asian and South Pacific Design Automation Conference (ASPDAC)*, pages 331–336, 2005.

[56] Gurobi Optimization. The gurobi optimizer 4.5. `http://www.gurobi.com/`.

[57] David Z. Pan, Peng Yu, Minsik Cho, Anand Ramalingam, Kiwoon Kim, Anand Rajaram, and Sean X. Shi. Design for manufacturing meets advanced process control: A survey. *Journal of Process Control*, 18(10):975–984, 2008.

[58] Min Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–55, 2005.

[59] D. Papa, N. Viswanathan, Z. Li C. Sze, G.-J. Nam, C. Alpert, and I.L. Markov. Physical synthesis with clock-network optimization for large systems on chips. *IEEE Micro*, 31(4):51–62, 2011.

[60] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov. Rumble: An incremental, timing-driven, physical-synthesis optimization algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(12):2156–2168, 2008.

[61] David A. Papa, Saurabh N. Adya, and Igor L. Markov. Constructive benchmarking for placement. In *ACM Great Lakes Symposium on VLSI*, pages 113–118, 2004.

[62] Ruchir Puri, Leon Stok, and Subhrajit Bhattacharya. Keeping hot chips cool. In *ACM/IEEE Design Automation Conference (DAC)*, pages 285–288, 2005.

[63] Vijay J. Reddi, David Z. Pan, Sani R. Nassif, and Keith A. Bowman. Robust and resilient designs from the bottom-up: Technology, circuit, cad and system issues. In *Asian and South Pacific Design Automation Conference (ASPDAC)*, 2012.

[64] H. Ren, D. Z. Pan, C. J. Alpert, P. Villarrubia, and G.-J. Nam. Diffusion-based placement migration with application on legalization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2158–2172, 2007.

[65] A.P.E. Rosiello, F. Ferrandi, D. Pandini, and D. Sciuto. A hash-based approach for functional regularity extraction during logic synthesis. In *IEEE Computer Society Annual Symposium on VLSI*, pages 92–97, 2007.

[66] A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut floorplacement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(7):1313–1326, 2006.

[67] Jarrod A. Roy and Igor L. Markov. Seeing the forest and the trees: Steiner wirelength optimization in placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(4):632–644, 2007.

[68] Jarrod A. Roy, David A. Papa, Saurabh N. Adya, Hayward H. Chan, Aaron N. Ng, James F. Lu, and Igor L. Markov. Capo: robust and scalable open-source min-cut floorplacer. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 224–226, 2005.

[69] Monica Sehrawat and Sukhvir Singh. Modified order crossover (ox) operator. *Computer Science & Engineering, Intl. Journal on*, 3(5):2019, 2011.

[70] P. M. Seidel. *On the Design of IEEIT Compliant Floating-point Units and Their Quantitative Analysis*. PhD thesis, University of Saarland, December 1999.

[71] T. Serdar and C. Sechen. Automatic datapath tile placement and routing. In *IEEE Design, Automation and Test in Europe*, pages 552 –559, 2001.

[72] Shelar and S. Rupesh. An algorithm for routing with capacitance/distance constraints for clock distribution in microprocessors. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 141–148, 2009.

[73] R. S. Shelar. A fast and near-optimal clustering algorithm for low-power clock tree synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(11):1781–1786, 2012.

[74] H. Shojaei, A. Davoodi, and J. Linderoth. Congestion analysis for global routing via integer programming. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 256–262, 2011.

[75] L. Sigal, J. D. Warnock, B. W. Curran, Y. H. Chan, P. J. Camporese, M. D. Mayo, W. V. Huott, D. R. Knebel, C. T. Chuang, J. P. Eckhardt, and P. T. Wu. Circuit design techniques for the high-performance cmos ibm s/390 parallel enterprise server g4 microprocessor. In *IBM Journal of Research and Development*, pages 489–503, 1997.

[76] P. Spindler, U. Schlichtmann, and F. M. Johannes. Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net

model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1398–1411, 2008.

[77] Wern-Jieh Sun and Carl Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):349–359, 1995.

[78] Cliff S. N. Sze. ISPD 2010 high performance clock network synthesis contest: benchmark suite and results. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, page 143, 2010.

[79] G. Venkataraman, Zhuo Feng, Jiang Hu, and Peng Li. Combinatorial algorithms for fast clock mesh optimization. *IEEE Transactions on Very Large Scale Integration Systems*, 18(1):131–141, 2010.

[80] N. Viswanathan, M. Pan, and C. Chu. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Asian and South Pacific Design Automation Conference (ASPDAC)*, pages 135–140, 2007.

[81] Natarajan Viswanathan, Charles J. Alpert, Cliff Sze, Zhuo Li, Gi-Joon Nam, and Jarrod A. Roy. The ISPD-2011 routability-driven placement contest and benchmark suite. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 141–146, 2011.

[82] Natarajan Viswanathan, Charles J. Alpert, Cliff C. N. Sze, Zhuo Li, and Yaoguang Wei. The dac 2012 routability-driven placement contest

and benchmark suite. In *ACM/IEEE Design Automation Conference (DAC)*, pages 774–782, 2012.

[83] Natarajan Viswanathan and Chris Chong-Nuen Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):722–733, 2005.

[84] Natarajan Viswanathan, Gi-Joon Nam, Charles J. Alpert, Paul Villarrubia, Haoxing Ren, and Chris Chu. RQL: Global placement via relaxed quadratic spreading and linearization. In *ACM/IEEE Design Automation Conference (DAC)*, pages 453–458, 2007.

[85] Jiří Šíma and Pekka Orponen. General-purpose computation with neural networks: a survey of complexity theoretic results. *Neural Comput.*, 15(12):2727–2778, 2003.

[86] Guosheng Wang. A survey on training algorithms for support vector machine classifiers. In *Networked Computing and Advanced Information Management*, pages 123–128, 2008.

[87] Maogang Wang, Xiaojian Yang, and Majid Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 260–263, 2000.

[88] Samuel I. Ward, Duo Ding, and David Z. Pan. Pade: A high-performance mixed-size placer with automatic datapath extraction and evaluation through high-dimensional data learning. In *ACM/IEEE Design Automation Conference (DAC)*, pages 756–761, 2012.

[89] Samuel I. Ward, Myung-Chul Kim, Nat Viswanathan, Zhuo Li, Charles Alpert, E. E. Swartzlander Jr., and David Z. Pan. Keep it straight: teaching placement how to better handle designs with datapaths. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 79–86, 2012.

[90] Samuel I. Ward, Myung-Chul Kim, Natarajan Viswanathan, Zhuo Li, Charles Alpert, Earl Swartzlander, , and David Z. Pan. Structure-aware placement techniques for designs with datapaths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(2):228–241, 2013.

[91] Samuel I. Ward, David Z. Pan, and E. E. Swartzlander Jr. 2011 ISPD Datapath Benchmark Suite.
http://www.cerc.utexas.edu/utda/download/DP/, 2011.

[92] Samuel I. Ward, David A. Papa, Zhuo Li, Cliff N. Sze, Charles J. Alpert, and E. E. Swartzlander Jr. Quantifying academic placer performance on custom designs. In *Proceedings ACM International Symposium on Physical Design (ISPD)*, pages 91–98, 2011.

[93] Samuel I. Ward, Natarajan Viswanathan, Nancy Y. Zhou, Cliff C.N. Sze, Zhuo Li, Charles J. Alpert, and David Z. Pan. Clock power minimization using structured latch templates and decision tree induction. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.

[94] J. Warnock, Yiu-Hing Chan, S. Carey, Huajun Wen, P. Meaney G., G. Gerwig, H.H. Smith, Chan Yuen, J. Davis, P. Bunce, A. Pelella, D. Rodko, P. Patel, T. Strach, D. Malone, F. Malgioglio, J. Neves, D.L. Rude, and W. Huott. Circuit and physical design implementation of the microprocessor chip for the zenterprise system. *IEEE Journal of Solid-State Circuits*, 47(1):151–163, 2012.

[95] Yaoguang Wei, Cliff Sze, Natarajan Viswanathan, Zhuo Li, Charles J. Alpert, Lakshmi N. Reddy, Andrew D. Huber, Gustavo E. Tellez, Douglas Keller, and Sachin S. Sapatnekar. Glare: Global and local wiring aware routability evaluation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 768–773, 2012.

[96] D.F. Wendel, J. Barth, D.M. Dreps, S. Islam, J. Pille, and J.A. Tierno. Ibm power7 processor circuit design. *IBM Journal of Research and Development*, 55(3):1–8, 2011.

[97] Joe G Xi and Wayne W.-M Dai. Useful-skew clock routing with gate sizing for low power design. In *ACM/IEEE Design Automation Conference (DAC)*, pages 383–388, 1996.

[98] Linfu Xiao, Zigang Xiao, Zaichen Qian, Yan Jiang, Tao Huang, Haitong Tian, and E.F.Y. Young. Local clock skew minimization using blockage-aware mixed tree-mesh clock network. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 458–462, 2010.

[99] Changqi Yang, Xianlong Hong, Yici Cai, Wenting Hou, Tong Jing, and Weimin Wu. Standard-cell based data-path placement utilizing regularity. In *IEEE International Conference on ASIC*, volume 1, pages 97–100, 2003.

[100] T.T. Ye, S. Chaudhuri, F. Huang, H. Savoj, and G. De Micheli. Physical synthesis for ASIC datapath circuits. In *IEEE International Symposium on Circuits and Systems*, volume 3, pages 365–368, 2002.

[101] Bei Yu, Xiaoqing Xu, Jhih-Rong Gao, and David Z. Pan. Methodology for standard cell compliance and detailed placement for triple patterning lithography. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.